M.Sc. (IT) Previous Year

MIT-08

VISUAL C++

मध्यप्रदेश भोज (मुक्त) विश्वविद्यालय – भोपाल

MADHYA PRADESH BHOJ (OPEN) UNIVERSITY - BHOPAL

Reviewer Committee

1. Dr. Sharad Gangale
   Professor
   R.K.D.F. University, Bhopal (M.P.)

2. Dr. Romsha Sharma
   Professor
   Sri Sathya Sai College for Women, Bhopal (M.P.)

3. Dr. Amit Kumar Mandle
   Assistant Professor
   I.E.H.E., Bhopal (M.P.)

Advisory Committee

1. Dr. Jayant Sonwalkar
   Hon'ble Vice Chancellor
   Madhya Pradesh Bhoj (Open) University, Bhopal (M.P.)

2. Dr. L.S. Solanki
   Registrar
   Madhya Pradesh Bhoj (Open) University, Bhopal (M.P.)

3. Dr. Kishor John
   Director
   Madhya Pradesh Bhoj (Open) University, Bhopal (M.P.)

4. Dr. Sharad Gangale
   Professor
   R.K.D.F. University, Bhopal (M.P.)

5. Dr. Romsha Shrama
   Professor
   Sri Sathya Sai College for Women, Bhopal (M.P.)

6. Dr. Amit Kumar Mandle
   Assistant Professor
   I.E.H.E., Bhopal (M.P.)

COURSE WRITERS

Dr. Vineeta Khemchandani, Associate Professor, Dept. of Information Technology, J.S.S. Academy of Technical Education, Noida, U.P.

Units (1.0-1.2, 1.5-1.7, 1.9-1.14, 2.4-2.4.1, 4.3, 4.6-4.6.1)

Dr. Preety Khatri, Assistant Professor, Computer Science, S.O.I.T. I.M.S., Noida

Units (1.3-1.4, 1.8, 2.0-2.3, 2.4.2-2.9, 3)

Sinchan Banerjee, Former Faculty, Dept. of I.T. and Management, Brainware University, Kolkata, W.B.

Units (4.0-4.2, 4.4-4.5, 4.6.2-4.11, 5)

Copyright © Reserved, Madhya Pradesh Bhoj (Open) University, Bhopal

All rights reserved. No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the Registrar, Madhya Pradesh Bhoj (Open) University, Bhopal

Information contained in this book has been published by VIKAS® Publishing House Pvt. Ltd. and has been obtained by its Authors from sources believed to be reliable and are correct to the best of their knowledge. However, the Madhya Pradesh Bhoj (Open) University, Bhopal, Publisher and its Authors shall in no event be liable for any errors, omissions or damages arising out of use of this information and specifically disclaim any implied warranties or merchantability or fitness for any particular use.

Published by Registrar, MP Bhoj (open) University, Bhopal in 2020

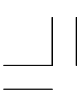
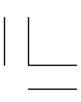
Vikas® is the registered trademark of Vikas® Publishing House Pvt. Ltd.

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

SYLLABI-BOOK MAPPING TABLE

Visual C++

Syllabi	Mapping in Book
UNIT - I **Windows Application Basics** : Windows and Windows Programming. **Visual C++ Basics**, Visual C++ and Windows Programming, Structure of a VC++ application, Starting VC++, A Sample VC++ (Win32) Application. **Dialogs and Controls**, Dialog boxes, Command Button Control, Check-box Control, Radio Button Control, List Box, Combo Box, slider Control, Messages, Message Queue, Handling Messages with Class Wizard. **Documents and Views**, The Documents Class, The View Class.	**Unit-1:** Basics of Windows Application and Visual C++ **(Pages 3-75)**
UNIT - II **Drawing on the Screen** : Device Contexts, Device Objects, Wizard Support for Device Context, Stock Objects, A DC Example, Using Color in Windows Applications. **Printing and Print Preview** : Printing, MFC Printing Application, Adding Functionalities to MFC Print **Persistence and File I/O** : File Basics, Files and Windows Applications, Serialization.	**Unit-2:** Drawing on the Screen, Printing and File Handling **(Pages 77-123)**
UNIT - III **Status Bars and Tool Bars** : Status Bar, Toolbars. **Common Controls** : Command Button Control, Check Box Control, Radio Button Control, List Box Control, Combo Box Control, Slider Control. **Help** : Building Blocks of help, **Property Pages and Sheets :** CpropertySheet, CpropertyPage	**Unit-3:** Status Bars, Tool Bars, Common Controls, Help, Property Pages and Sheets **(Pages 125-155)**
UNIT - IV **Common Controls :** ActiveX and OLE, ActiveX and COM, ActiveX and MFC, VC++ ActiveX Project, ActiveX Control Macros. **Building an ActiveX Container Application** : ActiveX Control Containers. **Building an ActiveX Server Application** : Component, Building and Using COM Server in VC++. **Building an ActiveX Control** : A Simple ActiveX Control Application, ActiveX Control Methods, ActiveX Events.	**Unit-4:** Common Controls **(Pages 157-224)**
UNIT - V **Socket, MAPI and the Internet** : Internet/Intranet Applications, Sockets, Ports and Addresses, Creating a Socket Program, Creating a Client Browser Program. **Internet Programming** : Create The Project, Set_MERGE_ PROXYSTUB, The Build Rule, **The Active Template Library** : An ATL Project. **Database Application** : ActiveX Data Objects, Creating a Database Application..	**Unit-5:** Internet Programming and Database Application **(Pages 225-284)**

CONTENTS

INTRODUCTION

UNIT 1 BASICS OF WINDOWS APPLICATION AND VISUAL C++ 3-75

1.0 Introduction
1.1 Objectives
1.2 Windows Basics and Visual C++ Program
1.2.1 Developer Studio Wizards
1.2.2 Developer Studio Display Area
1.2.3 Status Bar
1.2.4 Menu Bar
1.2.5 Using Visual C++ to Write Windows Program
1.3 Starting VC++ and Structure of a VC++ Application
1.4 A Sample VC++ (Win32) Application
1.5 Dialog Boxes
1.6 Controls
1.6.1 Button Control
1.7 Creating Controls
1.7.1 List Box
1.7.2 Combo Box
1.7.3 Slider Control
1.8 Messages
1.8.1 Message Queues
1.8.2 Handling Messages with Class Wizard
1.9 Document and Views
1.9.1 Classes for MDI Application
1.9.2 The View Class
1.10 Answer to 'Check Your Progress'
1.11 Summary
1.12 Key terms
1.13 Self-Assessment Questions and Exercises
1.14 Further Reading

UNIT 2 DRAWING ON THE SCREEN, PRINTING AND FILE HANDLING 77-123

2.0 Introduction
2.1 Objectives
2.2 Drawing on the Screen
2.2.1 Device Contexts
2.2.2 Device Objects
2.2.3 Wizard Support for Device Context
2.2.4 Stock Objects
2.2.5 A DC Example
2.2.6 Using Color in Windows Applications
2.3 Printing and Print Preview
2.3.1 MFC Printing Apple Cation
2.3.2 Adding Functionalities to MFC Print
2.4 Persistence and File I/O
2.4.1 Basic File Operations
2.4.2 Files and Windows Applications
2.4.3 Serialization

2.5 Answer to 'Check Your Progress'
2.6 Summary
2.7 Key Terms
2.8 Self-Assessment Questions and Exercises
2.9 Further Reading

UNIT 3 STATUS BARS, TOOL BARS, COMMON CONTROLS, HELP, PROPERTY PAGES AND SHEETS 125-155

3.0 Introduction
3.1 Objectives
3.2 Status Bars and Tool Bars
3.3 Common Controls
 3.3.1 Check Box Control
 3.3.2 Radio Button Control
 3.3.3 List Box Control
 3.3.4 Combo Box Control
 3.3.5 Slider Control
3.4 Building Blocks of Help
3.5 Property Pages and Sheets
3.6 Answer to 'Check Your Progress'
3.7 Summary
3.8 Key Terms
3.9 Self-Assessment Questions and Exercises
3.10 Further Reading

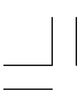
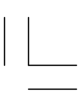
UNIT 4 COMMON CONTROLS 157-224

4.0 Introduction
4.1 Objectives
4.2 ActiveX and OLE
4.3 ActiveX and COM
 4.3.1 Creating MFC Project to Develop Car Component
4.4 ActiveX Control Macros
4.5 Building an ActiveX Server Application
4.6 Building ActiveX Control
 4.6.1 Creating ActiveX Control Container Application
 4.6.2 ActiveX Control Methods
 4.6.3 ActiveX Events
4.7 Answers to 'Check Your Progress'
4.8 Summary
4.9 Key Terms
4.10 Self-Assessment Questions and Exercises
4.11 Further Reading

UNIT 5 INTERNET PROGRAMMING AND DATABASE APPLICATION 225-284

5.0 Introduction
5.1 Objectives
5.2 Socket, MAPI and the Internet
 5.2.1 Creating a Socket Program
 5.2.2 Creating a Client Program
 5.2.3 Port
 5.2.4 Addresses
 5.2.5 Messaging Application Programming Interface (MAPI)

5.3 Internet Programming: Creating a Project
5.3.1 Set_MERGE_PROXYSTUB
5.3.2 The Build Rule - Understanding Custom Build Steps and Build Events
5.4 Active Template Library (ATL)
5.4.1 Creation of the Project
5.5 Database Application
5.5.1 ActiveX Data Objects (ADO)
5.5.2 Database Application using ADO
5.6 Answer to 'Check Your Progress'
5.7 Summary
5.8 Key Terms
5.9 Self-Assessment Questions and Exercises
5.10 Further Reading

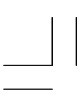
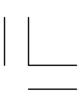
INTRODUCTION

NOTES

Microsoft Visual C++ (MSVC) is a 'commercial integrated development environment (IDE) product engineered by Microsoft for the C, C++, and C++/CLI programming languages'. It can develop and debug the C++ code for Microsoft Windows products. MSVC is a software that was originally a standalone product but later it is available in trialware and freeware norms when it becomes a part of visual studio.

Visual C++ is a fully functional framework for development. It is built on two important concepts. First, the development environment itself runs under Windows to provide a full set of Windows-based tools in order to create and manage the applications in Windows. Second, it uses the Visual user interface to handle the Windows based-tools. The Visual C++ environment is built around three elements: the C++ compiler and linker, the Developer Studio and the Microsoft Foundation Class Library. Visual C++ and Developer Studio are fully integrated with the environment to make it very easy to create Windows applications by using tools and Wizards provided as part of the Development Studio with an MFC class library.

This book, *Visual C++*, follows the SIM format wherein each Unit begins with an Introduction to the topic followed by an outline of the 'Objectives'. The detailed content is then presented in a simple and an organized manner, interspersed with 'Check Your Progress' questions to test the understanding of the students. A 'Summary' along with a list of 'Key Terms' and a set of 'Self-Assessment Questions and Exercises' is also provided at the end of each unit for effective recapitulation.

UNIT 1 BASICS OF WINDOWS APPLICATION AND VISUAL C++

NOTES

Structure

1.0 Introduction
1.1 Objectives
1.2 Windows Basics and Visual C++ Program
1.2.1 Developer Studio Wizards
1.2.2 Developer Studio Display Area
1.2.3 Status Bar
1.2.4 Menu Bar
1.2.5 Using Visual C++ to Write Windows Program
1.3 starting VC++ and Structure of a VC++ Application
1.4 A Sample VC++ (Win32) Application
1.5 Dialog Boxes
1.6 Controls
1.6.1 Button Control
1.7 Creating Controls
1.7.1 List Box
1.7.2 Combo Box
1.7.3 Slider Control
1.8 Messages
1.8.1 Message Queues
1.8.2 Handling Messages with Class Wizard
1.9 Document and Views
1.9.1 Classes for MDI Application
1.9.2 The View Class
1.10 Answer to 'Check your Progress'
1.11 Summary
1.12 Key terms
1.13 Self-Assessment Questions and Exercises
1.14 Further Reading

1.0 INTRODUCTION

Microsoft Visual C++ is a tool for building and debugging Window-based applications and libraries in an Integrated Windows environment. Visual C++ makes it much easier to handle the complex job of developing applications for Windows by incorporating on integrated Windows-based environment with high-level C++ application classes.

Visual C++ is a fully functional framework for development. It is built on two important concepts. First, the development environment itself runs under Windows to provide a full set of Windows-based tools in order to create and manage the applications in Windows. Second, it uses the Visual user interface to handle the Windows based-tools.

The Visual C++ environment is built around three elements: the C++ compiler and linker, the Developer Studio and the Microsoft Foundation Class Library.

The structure of Visual C++ Application consists of the basic structure of a VC++ Application, where first you should start Visual C++ on your computer so that you can see about the area that how each of the areas are arranged or how you can alter the arrangement and make changes in that area.

In many Windows-based applications, controls are placed in the dialog box rather than directly on the application's window because creation and use of controls is much simpler on the dialog box than on the window. Dialog boxes are used to enter the user's input to the application through controls like text box, button control, edit box, selecting options, checking values, etc. Another use of dialog boxes is in the validation of data and the data exchange between the controls. There are different ways in which Windows applications can receive the user's inputs. It can be either through a keyboard, through a mouse or by selecting a menu item. In case of a keyboard input, a message is generated and sent to the Window, which has input focus, and in case of a mouse input the message is sent to the window that is currently under the cursor. Other than these conventional input methods, Windows application can take input from the text box, edit control, push button, list box and combo box. This unit includes various user interface controls. These controls can appear either on the applications windows or on the dialog box. When these controls are needed to be displayed on the dialog box, they are not required to be created explicitly but can be defined during the dialog box creation.

An MDI application is based on objects of two main classes, a mainframe window class object and many child window class objects. This also includes multiple type view classes, which an MDI application can hold to represent different views of the same data. This unit introduces multiple document interface, which is a standard way to write application in which one master window holds a number of child windows.

In this unit, you will learn about the windows basics and visual C++ program, starting VC++ and structure of a VC++ application, a sample VC++ (Win 32) applications, dialog boxes, controls, messages and documents and views.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basics of Windows and Visual C++ program
- Learn about the starting VC++ and structure of a VC++ application
- Explain about the sample VC++ (win 32) application
- Understand and design a dialog box
- Create controls using create () function
- Define the various types of button controls
- Discuss about the messages and their significance
- Elaborate on the MDI and view class

1.2 WINDOWS BASICS AND VISUAL C++ PROGRAM

NOTES

The Microsoft Developer Studio is the centre of the Visual C++ development environment. It is used to integrate the development tools and the Visual C++ compiler. A Windows program can be created, scanned through an impressive amount of online help and debugged without the Developer Studio.

Visual C++ and Developer Studio are fully integrated with the environment to make it very easy to create Windows applications by using tools and Wizards provided as part of the Development Studio with an MFC class library.

Developer Studio Tools

Developer Studio tools are used for complete project management. They are used to link a variety of code modules into one project, which is then used as a unit for building the applications.

1. Developer Studio Code Editor

Code editor is used to edit the C++ source code file that will be compiled into the Windows program.

The Developer Studio editor is similar to the word processor, but it not only provides text formatting but also provides features that help to write the source code easily.

Features of the Code Editor

The features of code editor are as follows:

- Automatic syntax highlighting shows keywords, comments and other source codes in different colors.
- Automatic 'Smart' indenting helps line-up the code into easy-to-read columns.
- Integrated keyword help enables to get help on any keyword, MFC classes or windows functions, just by pressing F1.
- Drag-and-drop editing enables to easily move the text by dragging it with the mouse.
- Integration with the compiler's error output helps step through the list of errors reported by the compiler and position the cursor at every error.

2. Resource Editor

Microsoft Developer Studio Resource editors share techniques and interfaces to create and modify application resources. They can create all Microsoft Windows resources, such as:

- Accelerator Tables
- Bitmaps
- Cursors
- Dialog Boxes
- Icons

- Menus
- String Tables
- Tool Bars

NOTES

When any resource is created or opened, the corresponding editor opens automatically. Graphical resources, such as toolbars, cursors, icons, are negotiable bitmaps. The accelerator table and string tables are formatted text. Dialog boxes are a combination of both the graphical and the text resource.

3. Integrated Debugger

An integrated debugger enables the program to check errors. Debugger is a part of the Developer studio. It finds and corrects bugs. If an error occurs while debugging, the source code can be corrected and compiled.

1.2.1 Developer Studio Wizards

In addition to the tools that are used for creating, editing and debugging resources, the Developer studio includes two wizards to develop Windows applications.

1. Appwizard

AppWizard is used to create the basic outline of the Windows application. It supports three types of applications. Single Document and Multiple Document applications are based on the document view architecture. In Dialog box-based applications, the dialog box serves as the application's main window.

2. Class Wizard

ClassWizard is used to define the class in a program created with the AppWizard. Using the ClassWizard, class can be added to the project. The ClassWizard is also used to add functions that control how the messages received the by each class are handled.

The ClassWizard also helps to manage the controls that are contained in dialog box by associating an MFC object or class member variable with each control.

Microsoft Foundation Class (MFC) Library

The MFC enables us to write Windows application using C++. The class library consists of C++ classes that represent an application frame work.

The classes are designed to be used together to create a working skeleton application that provides much of the user interface and functionality that the users expect in a Windows application.

Functionality of the skeleton program can be extended by simply overloading the appropriate classes with new ones that only need to provide the new functions that are required.

The visual C++ environment is directly linked into the MFC by two wizards: AppWizard and ClassWizard.

Standard Template Library

A recent addition to the C++ draft standard is the Standard Template Library (STL). Unlike the MFC class library which is used primarily for Windows Programming, the STL is used for general-purpose programming using the template.

InfoViewer

InfoViewer is the outline help system, integrated into the Developer Studio.

InfoViewer is the only documentation included with the product because Visual C++ is not sold with a documentation.

InfoViewer has several advantages over a hard copy of documentation, such as:

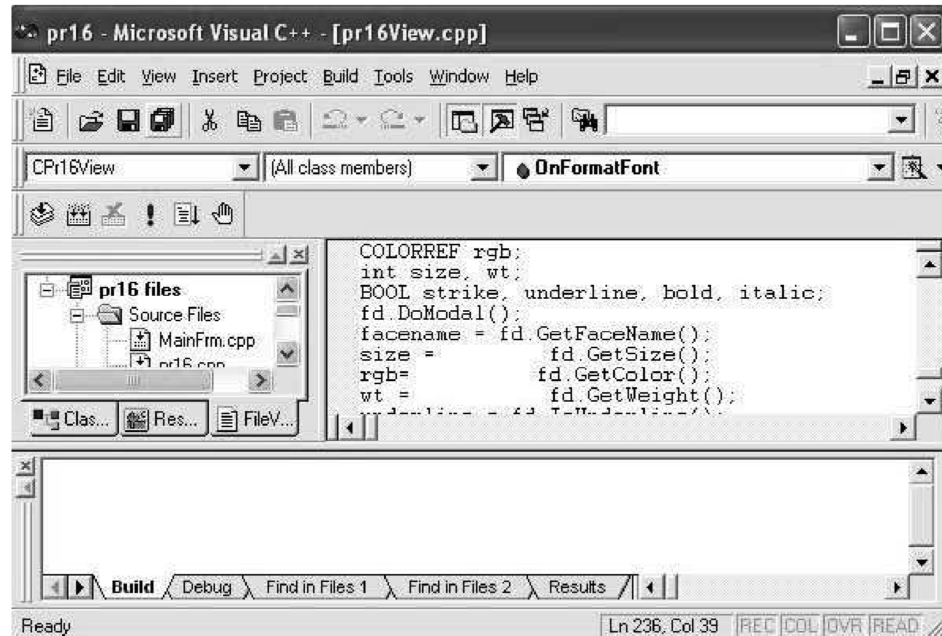
- It is fully searchable.
- Annotations and bookmarks can be added in the documentation.
- Context sensitive help can be used pressing the F1 key.
- It is completely integrated into Developer Studio. One of the tabs in the project workspace window displays the InfoViewer table of contents.
- InfoViwer documentation can be printed where a hard copy is required.

1.2.2 Developer Studio Display Area

The default display of Developer Studio environment consists of

1. Standard Window title bar with the usual control boxes shows the name of the current project, if one is opened.
2. The menu bar.
3. The standard Toolbar.
4. The Project Toolbar.
5. On the left side of the main display is the project workspace. Project Workspace displays the various types of information about the open project and the InfoViewer hierarchy of help information.
6. On the right side of the main display is the Source Editor pane, Source editor displays the files opened for editing and any other Developer Studio Windows that are activated or user activate.
7. Beneath the top two panes is the output pane. An output pane displays a variety of output information from the build and debug processing, as selected by the tabs at the bottom of the pane.
8. The Status Bar.

NOTES



Toolbars

A toolbar is a control bar that contains buttons that can have the push-button, checkbox or radio button style. It is usually aligned at the top of a frame window. A toolbar's typical purpose is to provide an alternative interface for the menu commands.

The Developer Studio has eight standard toolbars. The display and modification of these toolbars can be controlled by selecting the Toolbars command from the View menu. Two of the eight toolbars, standard and project are displayed by default, while other toolbars are displayed by default at the appropriate times, for e.g. the dialog toolbar is displayed while debugging the application.

1.2.3 Status Bar

The Status bar, which is displayed along the bottom of Developer Studio window by default, provides valuable information about the current state of the Developer Studio and files.

The status bar has several panes that display information about the current status of the project.

When using the editor, the default panes display the following information, from left to right.

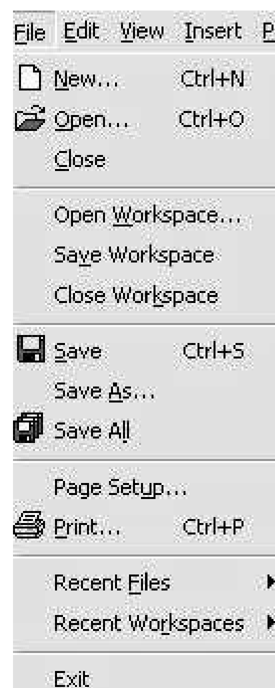
- The Help Message pane displays a short help message about the selected function or a message that tells about actions or errors that have happened.
- The Cursor Location pane shows the number of lines and columns the insertion point has set. Lines and columns are numbered from 1, if the range is selected, the number shown is the number of the line and / or column after the last selected line.
- The Macro Recorder pane displays REC while recording keystrokes for a macro. Macro recording is enabled electing record keystrokes from the Tool Menu.

- The Column Select pane displays COL, If selection mode is column. The column selection mode allows to select columns rather than lines of text. To enter the column select mode, hold down the Alt key and drag the cursor to the bottom right corner of the text. The resulting rectangular block of the text is selected. This is the column select mode.
- The Overtype pane displays the OVR if the Developer Studio editor is in over-type mode. The pane is dimmed if the editor is in insert mode (the default). This setting is toggled by pressing the insert key on the keyboard.
- The Read Only pane displays READ if the current file is a read-only file. It is dimmed if the file can be edited and saved in the normal way.
- The Clock pane displays activated in the Workspace tab of the Options dialog box.

1.2.4 Menu Bar

The menu bar gives access to all the functions in the Developer Studio. There are few functions that are not directly accessible from the menu selection.

The following screenshot displays the File Menu.



The File Menu has the following commands.

- **New:** Displays the New dialog box which allows to create a new version of any one of several types of files.
- **Open:** Displays the Open file dialog box which allows selecting a file to be opened for editing. The opened file becomes the current file displayed on the Developer studio.
- **Close:** Closes the current working file, if no working file is available or if the cursor is not positioned in a file that can be closed, this entry is dimmed.

NOTES

- **OpenWorkspace:** Displays the Open File Dialog Box, which allows to select a workspace file(.MDP) to be opened. This closes any project workspace currently open and opens the selected project, which becomes the current project in the Developer Studio.
- **Close Workspace:** Closes the current project workspace.
- **Save:** Saves the current file. This entry is available only if the current file has been changed.
- **Save As:** Displays the Save As dialog box, which allows saving the current file under the new name.
- **Save all:** Saves all changed files that are currently opened.
- **Find in files:** Displays the find in files dialog box which allows to search for text in a set of files or folders. This is a powerful and sophisticated searching tool, similar to the UNIX grip command that allows to find items across many files.
- **Page Setup:** Dispalys the Page setup dialog box which allows to set margins for page and adds header and footer text.
- **Print:** Dispalys the Print dialog box, which allows to print the current file using the currently slected printer and print settings.

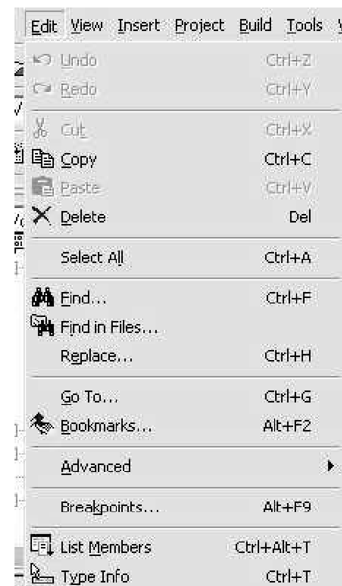
The next section of the File Menu displays the last four files that are opened. The user may select any one of them to immediately open and make it a current file.

The next section of the File menu displays the last four projects that are opened.

- **Exit:** Closes all open files and exits the Developer Studio. If any open file has changes and is not saved, it prompts to save it.

View Menu

The View menu controls what to see on the display. The following screenshot displays the View Menu.



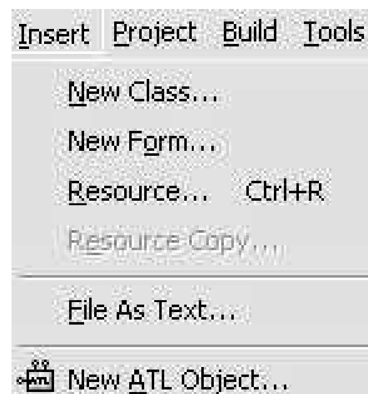
View menu contains the following commands.

- **ClassWizard:** Takes the user to the ClassWizard, which allows to manage classes.
- **Resource Symbols:** Displays the Resource Symbols dialog box, which displays all the symbols currently defined for the project in the Resource File with the assigned value and whether the resource is in use within the project.
- **Resource Includes:** Displays the Resource includes dialog box, which shows the header files that are included in the resource file.
- **Full Screen:** Toggles the edit pane to full screen display.
- **Toolbars:** Displays the Toolbars dialog box to display or hide selected toolbars.
- **InfoViwer:** InfoViewer History list shows the last 50 InfoViewer topics.

The next section of the View menu displays all the panes of the Developer Studio.

Insert Menu

The Insert menu allows to insert items into the project. The following screenshot displays the Insert Menu:



Commands in the Insert menu are:

- **File:** allows selecting and inserting an existing file into a currently active file at the current insertion point.
- **Resource:** allows inserting a new resource into the project.
- **Resource Copy:** Inserts a copy of the selected resource into application.
- **Files into Project:** Displays the Insert Files into Project dialog box. This allows to add new files to the project, which will then be included in the project's next build.
- **Project:** Allows to insert an existing project into a current project as a unit.

Component: Component allows to select a component from the Component gallery to insert into project.

NOTES

Build Menu

The following screenshot display the Build menu.

The Build Menu has the following commands.

- **Compile file:** Compiles the current file. The current file's name is displayed to the right side of the command.
- **Build Application:** Builds the application represented by the current project. The name of the executable file that will be generated is displayed next to the command. Build compiles or links only these modules that have been updated since the last time the application was created. If the application is up-to-date the user will see a dialog box to show that the application is ready to run.
- **Rebuild All:** Builds all components of the current project into the executable file whose name is displayed next to the command. It ignores whether the files have been changed.
- **Batch Build:** Displays the Batch dialog box, which allows to build the selected target, version of application , either building only out-of-date components or all the components. This feature allows to build both the debug and the release version..
- **Stop Build:** Stops the build process. Note that the keyboard shortcut of this command and of the Build command are identical. If the user is in build mode, using keyboard B key will stop building.
- **Update All dependencies:** Scans all the files in the project for dependencies and relinks the dependent files in the project.
- **Debug:** Starts the current application using the debugger, if the current application is not up-to-date, it presents the same warning dialog box as displayed white selecting the execute command.
- **Execute Application:** Executes the application file which is displayed on the right of the command.

- **Settings:** Displays the project settings dialog box, which allows to set a variety of important variables for the project.
- **Configuration:** Displays the configurations dialog box, which allows to create, delete, and manage projects and configurations to the current project workspace.
- **Subprojects:** Displays the subprojects, dialog box which allows to create and manage the subprojects of the main project. A subproject is a logically separate set of code that is included into the project as a unit.
- **Set default Configuration:** Displays the default project configuration dialog box, which allows to select one of the current configurations.

Help Menu

The following screenshot displays Help menu.

Each of the Help menu items allows to access the help files.

Structure of Project Workspace

The Project Workspace controls which files are included in the application and govern how they are combined to build the finished application.

- It stores the types of applications that are created.
- It allows the Developer Studio to keep track of all the elements that go into the application.
- It allows the facility to compile and link only those modules that have changed since the last time the project was built.
- It also stores all the compiler and linker settings, so it is not required to reset them every time the project is reloaded.

Elements of Project Workspace

Project Workspace consists of a project subdirectory, which contains at least two files.

1. Make file with extension .MAK.
2. Workspace file has extension .MDP

The project directory is the root directory for all the work done in the project. It usually contains the source files for top level projects, but files can be

added from any accessible location to the project workspace. The make file defines the rule that must be followed by the compiler and linker to build an application.

NOTES

The make file lists all the files involved in the creation of the final application, including libraries, resource files and included headers.

The project workspace file is used only by the Developer Studio and controls things such as the arrangement of panes and windows in the workspace.

A project workspace contains two special elements.

(i) **Configuration:** A configuration is a collection of settings for a project that defines what platform an application will run on and the tool settings for building the application.

(ii) **Project:** A project is a set of source files that defines the code, along with an associated configuration that defines the type of application to be build.

A project workspace may contain multiple projects and projects may themselves contain sub-projects.

Every project workspace contains one top-level project on which all the other projects and sub-projects depend.

Windows Program

In Windows environment everything is shared—the screen, the keyboard, the mouse, even the user. Program written for Windows must cooperate with Windows and with other programs that may be running at the same time. Windows program differs from most of the sequentially executed programs that assume complete control over all.

All resources in corporative environments like Windows, messages are sent to a program when an event occurs that affects the program. Every message sent to a program has a specific purpose. For e.g. Messages are sent when window is created; a menu item is selected etc. Responding to event messages is a key part of most Windows programs. Windows program must request the operating system for resources before use and once used must be returned to the operating system so that they can be used by the other programs.

In this way, Windows controls access to resources like the screen and other physical devices. Every Windows application that confirms to the Windows standards provides the following features:

- Every running application has a primary application window that displays the name of the application and the primary application menu bar.
- An application may support multiple documents at once. It is called 'Multiple Document Interface or (MDI)', and each document that is open is displayed in one or more document windows.
- Some applications such as Write, Notepad, which come with Windows only allow to have one document open at a time. This is called 'Single Document Interface (SDI)'.

1.2.5 Using Visual C++ to Write Windows Program

NOTES

When using Visual C++, much of the complexity of writing a Windows program is easily handled. The integrated tools used to create a program can make the job much easier by writing much of the required source code and by taking advantage of the MFC class library.

The following parts of the Visual C++ package simplify the writing program for Windows.

- **The actual C and C++ compiler:** C and C++ compiler can also be used to write program taking advantages of facilities provided by the Developer Studio, like syntax highlighting, integrated help and debugger.
- **The MFC class library Version 4.0:** MFC class library includes new classes that enable to add new features with just a few lines of code.
- **The ClassWizard and AppWizard:** The wizards that are included as a part of the Developer Studio enable to get started writing the Windows program by generating a skeleton application.

Check Your Progress

1. How is Microsoft Developer Studio useful?
2. Name the Developer Studio Tools.
3. Name the applications supported by AppWizard.
4. What is the function of Microsoft Foundation Class?
5. State the advantages of InfoViewer over hard copy documentation.
6. Which function does the command OpenWorkspace perform?
7. What is the function of Project Workspace?

1.3 STARTING VC++ AND STRUCTURE OF A VC++ APPLICATION

The structure of Visual C++ Application is defined by its control flow in the developer studio. You should start Visual C++ on your computer so that you can see the area and understand how each of the area is arranged or how you can alter the arrangement and make changes in that area. When the Microsoft Visual development environment starts, it will displays a window area as shown in Figure 1.1. In the Developer Studio environment, each of the areas has a particular purpose, so you can rearrange these areas also. You can customize the Developer Studio environment as per the specific development needs.

After starting the Visual C++, you can see the area on the left side of Developer Studio and that area is known as the workspace. Here, you can do the navigation of the various parts and pieces of your development projects. The parts of your application in that particular workspace can be viewed in three different ways:

NOTES

1. ***Resource View:*** This view allows you to edit as well as to find the various resources in your application. It consists of various icons, menus, dialog window designs, etc.
2. ***File View***: This view allows you to display the file views which are associated with your application. It also navigate all the files that make up your application.
3. ***Class View:*** The class view displays the class level view of your source code. It allows you to manipulate and navigate the source code also.

Fig. 1.1*Window Area in the Developer Studio Environment*

The Editor Area

The right side area of the Developer Studio environment is the editor area. While using Visual C++, you can use this area to perform all your editing. When you edit the C++ source code, the code editor Windows will display it. The window painter displays a dialog box when you design it. When you design the icons in your application, then the icon painter displays in the editor area. So, the editor area covers the whole Developer Studio area that is not otherwise occupied by menus, toolbars or panes.

The Output Pane

After compilation of the first application, the output pane displays at the bottom of the Developer Studio environment. It is not visible when you start Visual C++ for the first time. This pane remains open until you close it. In this pane, the Developer Studio displays the information like the compiler progress statements, error messages or warnings etc. The Visual C++ debugger shows all the variables having values as you step over your code. After closing this pane, Visual C++ reopens itself if it has any message that needs to display for you.

Menu Bars

You can see the three toolbars just below the menu bar. There are various other toolbars which are also available and depending on your interest you can customize and create your own toolbars. The three main toolbars are as follows:

NOTES

1. **The Build minibar** provides the build and run commands which are used to develop and test your applications. It is also used to switch between multiple build configurations.
2. **The Wizard Bar** toolbar offers to execute a number of Class Wizard actions without opening the Class Wizard.
3. **The Standard toolbar** is used for editing various applications. It consists of various standard tools for opening files, saving files, pasting, copying, cutting etc.

Rearranging the Developer Studio Environment

The Developer Studio provides the easiest methods to rearrange your development environment. This can be performed by right-clicking the mouse over the toolbar area. Figure 1.2 displays the pop-up menu that lets you to turn on/off various panes and toolbars.

Fig. 1.2 Toolbar Menu Area

Creating the Project Workspace

In C++, to create the project workspace, each application development requires its own project workspace. The project workspace consists of directories in which the application source code is set aside, and the directories in which several build configuration files are placed. The new project can be created with the help of following steps:

NOTES

1. Select File → New. This will opens the New Wizard as shown in Figure 1.3.

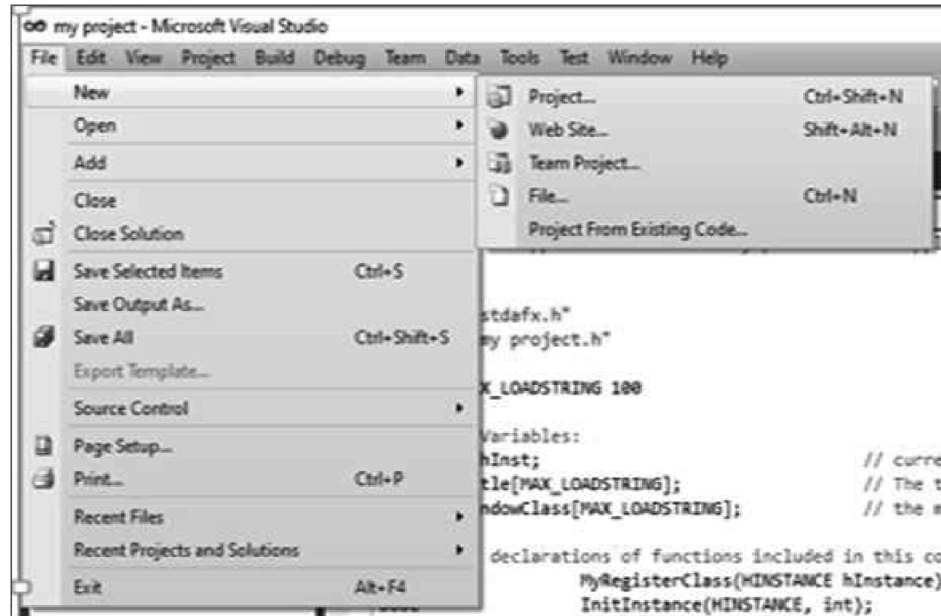


Fig. 1.3 Opening New Wizard

After opening new project, click on Visual C++, go to MFC Application as shown in Figure 1.4.

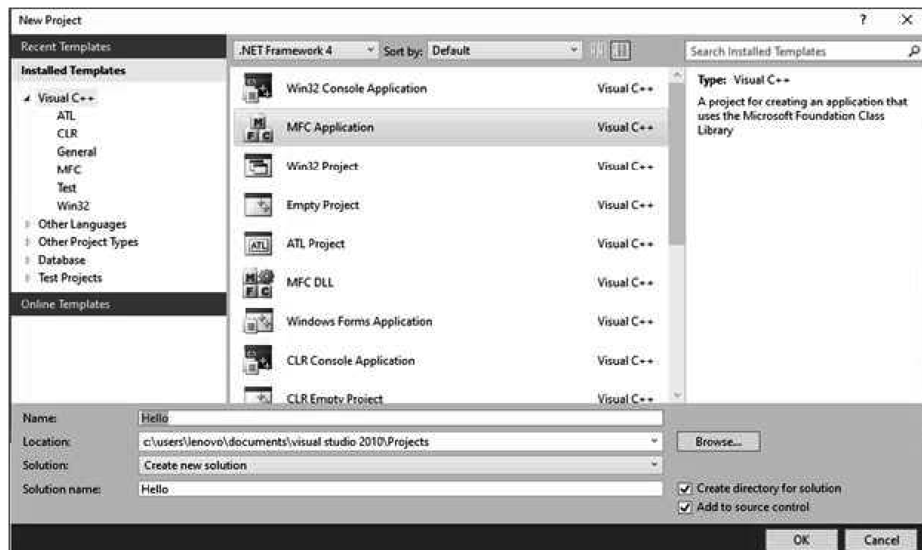


Fig. 1.4 Opening New MFC Application

2. On the Projects tab, select MFC AppWizard (exe).
3. Then type a name for your project, in the Project Name field, like Hello.
4. Click OK. It will open the New Wizard to create a project directory at a specific location and after that start the AppWizard as shown in Figure 1.5.

NOTES

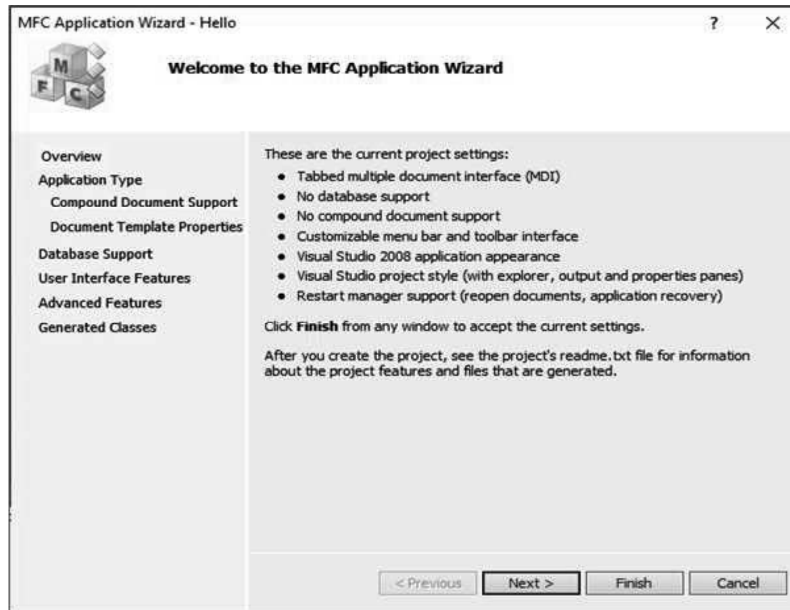


Fig. 1.5 *Starting the MFC Application Wizard*

Creating the Application Shell using Application Wizard

The AppWizard has very different functionality where it asks a series of questions like what features and functionality you require or type of application you are building etc. this information is used to create a shell of an application that can be compiled and run immediately. The application shell provides the basic infrastructure which is required to build the application. The steps to build an application are:

Step 1: Identify the type of Dialog-based application you want to create. Then, click Next which is at the bottom of the wizard.

Step 2: Here the AppWizard examines the number of features which is required in the application.

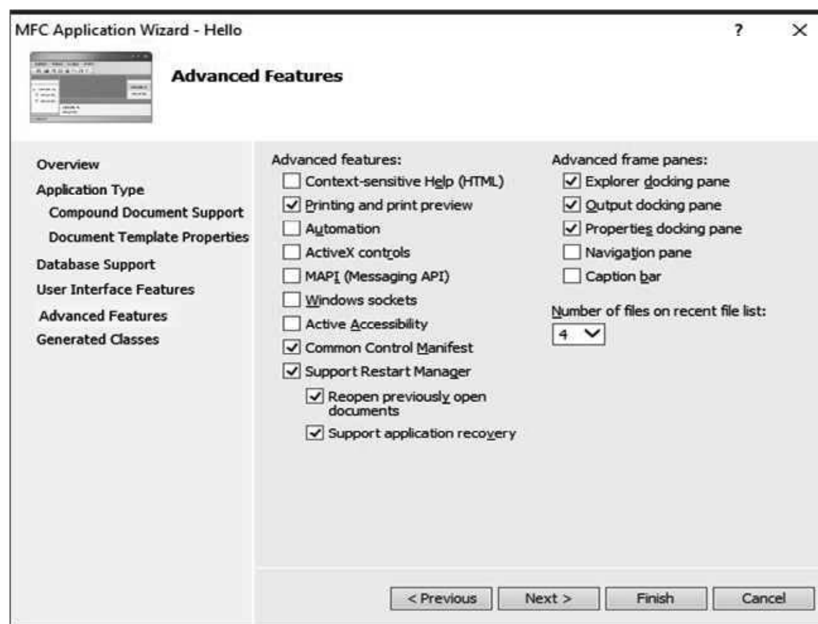


Fig. 1.6 *MFC Application Wizard*

NOTES

Step 3: Near the bottom of the wizard, first delete the project name which you named as "Hello" and write the title which you want to display in the title bar of the main application Window, and then click Next.

Step 4: Click Next at the bottom of the wizard to continue to the next step.

Step 5: This is the final step which displays the C++ classes that the AppWizard will produce for your application and then to generate the application, click Finish.

Step 6: Before AppWizard creates your application shell, it presents you with a list of what it is going to put into the application shell, based on the options you selected when going through the AppWizard. Click OK and AppWizard generates your application.

Step 7: After the creation of application shell, the workspace pane displays the tree view of the classes in your application shell, as shown in Figure 1.6.

Step 8: Select Build → Build Hello.exe to compile your application.

Step 9: After compilation of application, the VC++ compiler builds the application, and the output pane shows the message that displays the warnings and errors, as shown in Figure 1.7

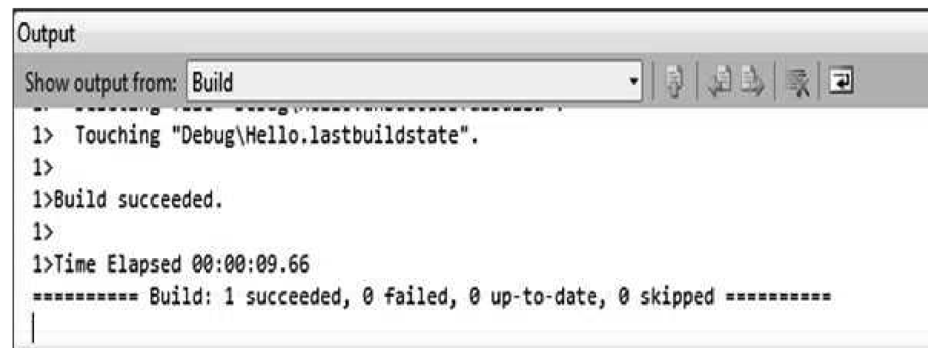


Fig. 1.7 *Building the VC++ Application*

Step 10: Select Build'! execute Hello.exe to run your application.

Step 11: Now, the application shows a dialog with a TODO message, OK and Cancel buttons. You can click the OK button to close the application.

1.4 A SAMPLE VC++ (WIN32) APPLICATION

To create a VC++ Project for a Win32 Console Application, first select File '!New'! Project as shown in Figure 1.8.

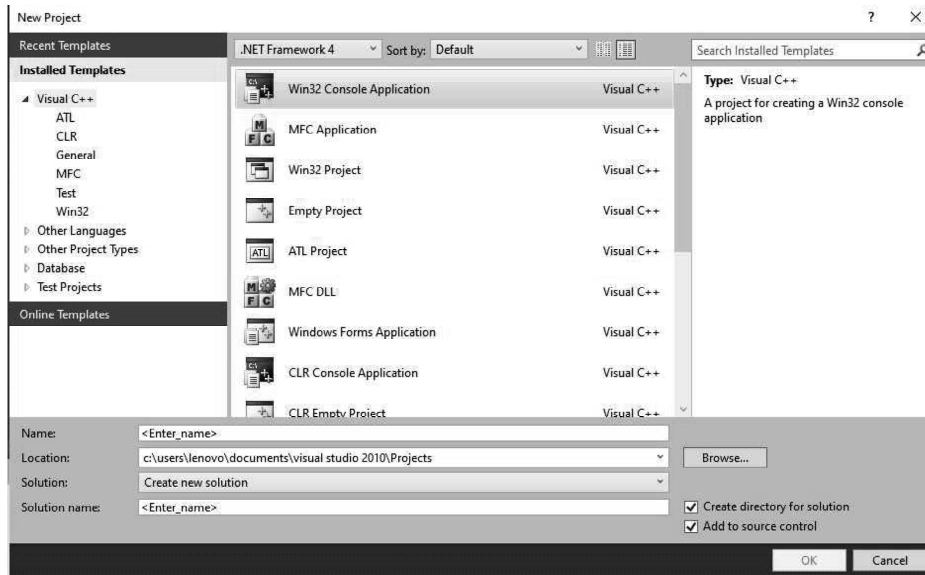


***Fig. 1.8** Sample VC++ Application*

Following are steps to create the VC++ (Win 32) Application.

1. In the New Project dialog box, there is a left pane which shows the types of projects you can create. You have to click on Win32 that shows the Application Wizard which is used to create the initial contents for the project.
2. Here, the right pane shows the list of templates which are available for the project type you have selected in the left pane. While creating the files, the template selected is used by the Application Wizard that makeup the project.
3. For customizing the files, there is the dialog box where you have to click on the OK button. Most of the template options have a basic set of program source modules which are created automatically.
4. Select an appropriate name for your project for example My Project2, or you can choose your own project name. VC++ supports the long filenames also. By default, the solution folder has the same name as the project and solution folder name displays in the bottom edit box and it can be changed as per your requirement.
5. To modify the location for the solution, there is a dialog box with help of it you can edit the location of your project. If you enter a name for your project, then the solution folder is by default set to a folder with that name as well as with the path shown in the Location edit box.
6. Use the Browse button to select another path for the solution. To specify a different path for the solution folder, you have to enter it in the Location edit box.
7. On clicking the OK button shows the Win32 Application Wizard dialog box as shown in Figure 1.9.

NOTES

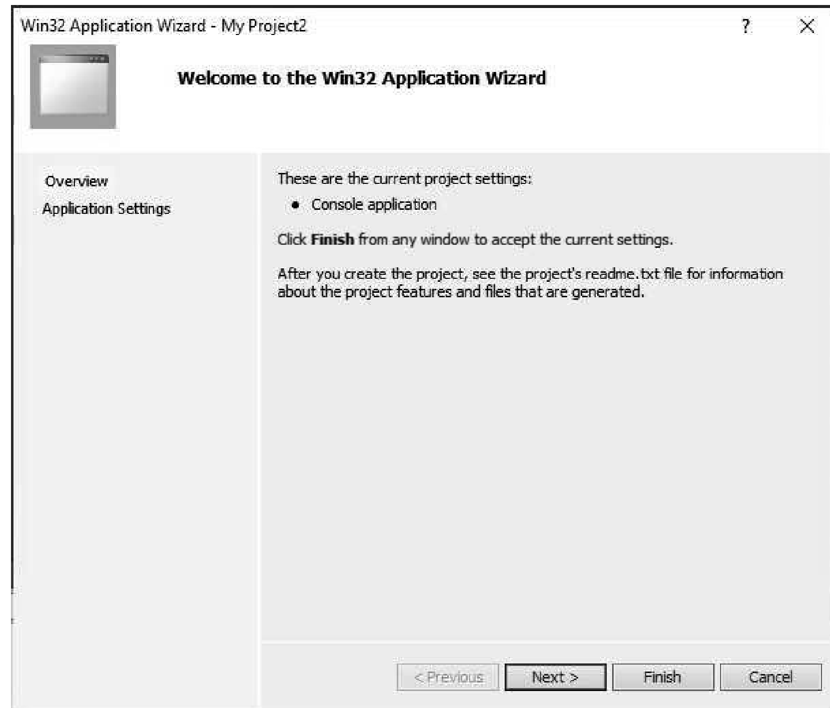


Fig. 1.9 *Win32 Application Wizard Dialog Box*

After clicking the Finish button, the wizard generates all the project files. You have to click on the Application Settings tab to display its settings as shown in Figure 1.10. This page lets you to select options which you want to apply to the project. You can select the Empty project checkbox when you are learning the C++ language but here you can leave things as they are and click the Finish button. The Application Wizard then produces the project with all the default files.

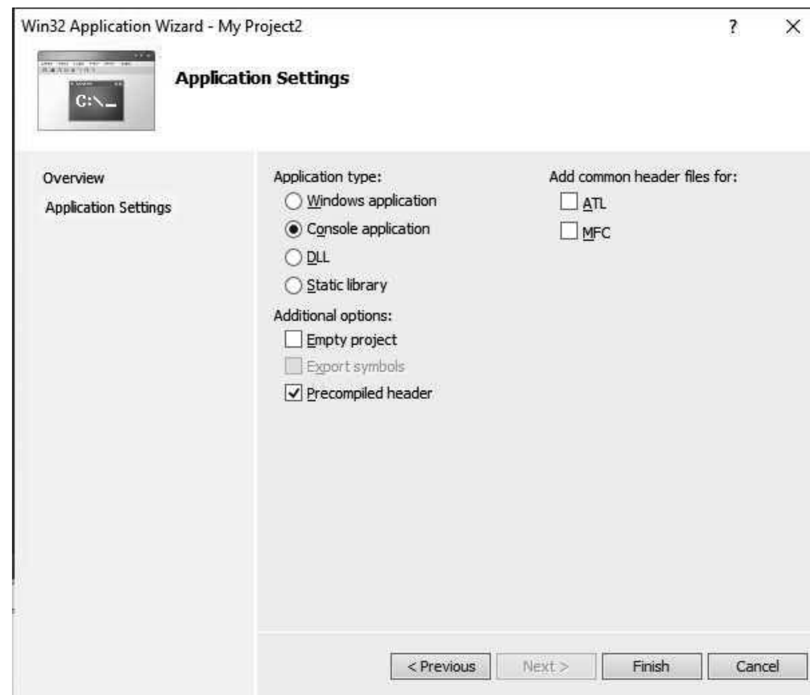


Fig. 1.10 *Win32 Application Settings Wizard*

NOTES

The project folder having the project name that you suggested. This will contain all the files which are making up the project definition. The solution folder has the same name like the project folder, in case if it's not changed. It consists of the project folder and the files which are describing the contents of the solution. To inspect the contents of the solution folder, if you use Windows Explorer, then it will display three files:

- A file having an extension .ncb that records data about **Intellisense.** Here the Intellisense is the ability that offers prompting and auto-completion for code in the Editor Window as you enter it.
- A file having extension .suo that records the user options applied to the solution.
- A file having extension .sln that records information about the projects in the solution.

If you are using Windows Explorer, then there are seven files primarily, comprising of a file with the name ReadMe.txt that holds a summary of the contents of the files that have been produced for the project. The Solution Explorer tab is shown in Figure 1.11 that shows a view of all the projects in the current solution as well as the files also. By double-clicking in name of the Solution Explorer tab, you can display the contents of any file as an additional tab in the Editor pane. You can switch immediately between any of the files also, that have been displayed by clicking on the particular tab.

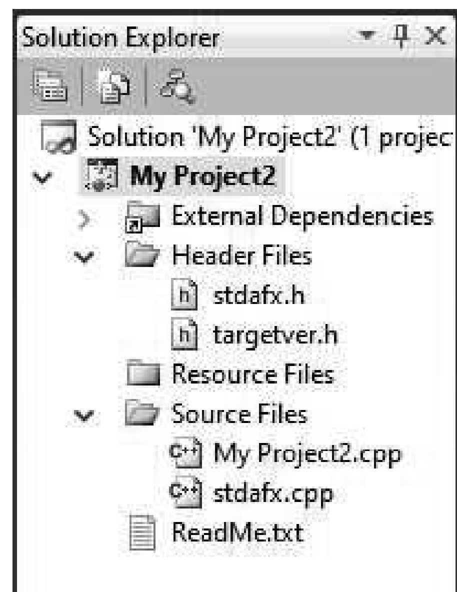


Fig. 1.11 *Solution Explorer Tab*

The ***Class View*** tab shows the classes defined in your project. It also displays the contents of each class. By default, the view is empty, it means it doesn't contains any classes in this application.

The ***Property Manager*** tab displays the properties that have been set for the Debug and Release versions about the project. By right-clicking a property and selecting Properties from the context menu, you can change any of the properties also. By pressing the Alt and F7 button, it shows the properties dialog box.

The ***Resource View*** displays the dialog boxes, menus toolbars, icons and other resources also which are used by the program. With the help of this tab, you can edit or add the resources easily.

NOTES

When you right-click items displayed in the tab, the Solution Explorer and other tabs offers context sensitive pop-up menus. While writing code, if the Solution Explorer pane gets in your way, then you can hide it by clicking the Auto hide icon and if you want to redisplay it then click the name tab on the left of the IDE window.

Modifying the Source Code

Sometimes, you program doesn't work as per your objectives. So, you need to change the code based on the requirements. In the Solution Explorer pane, double-click on My Project2.cpp. This file is the main source file for the program that the Application Wizard created as shown in Figure 1.12.

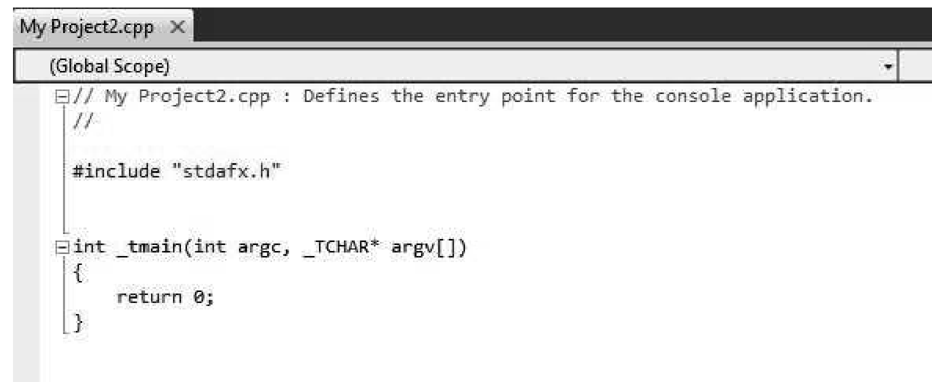


Fig. 1.12 *Modifying the Source Code (File name: My Project2.cpp)*

You can select Tools'! Options from the main menu to exhibit the Options dialog box, if the line numbers are not displayed on your system. Go to option Text Editor subtree in the right pane and then select General from the extended tree. Figure 1.13 shows the source code. Here first two lines are comments and anything following double forward slash (//) in a line is ignored by the compiler. In this code, Line 3 is #include directive which adds the contents of the file stdafx.h to this file in position of this #include directive. This is the way of adding the contents of .h source files to a .cpp source file a in a C++ program.

In this code, Line 5 is beginning of the function _tmain () and first line of the executable code in this file. In a C++ program, a function is simply a named unit of executable code and every C++ program have atleast one function. Here Lines 6 and 9 consists of left and right braces, and these encompass all the executable code in the function_tmain(). You can add this code to the Editor Window.

```
// Ex1_01.cpp: Defines the entry point for the console
application.
//
#include "stdafx.h"
#include <iostream>
int _tmain(intargc, _TCHAR* argv[])
{
```

```
std::cout<< "Hello World!\n";
return 0;
}
```

NOTES

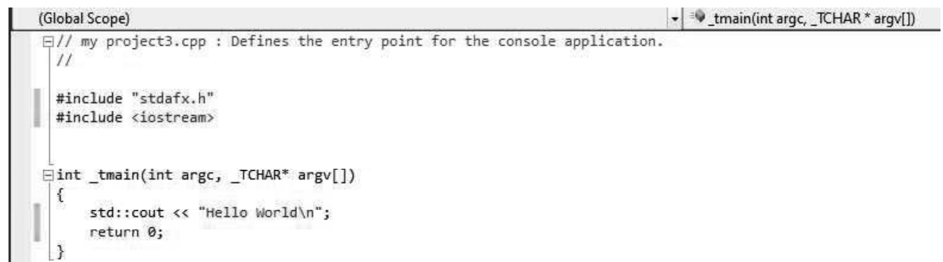


Fig. 1.13 Source Code

In this code, #include directive adds the contents of one of the standard libraries for VC++ to the source file. The <iostream> library defines facilities for basic I/O operations and second line writes output to the command line. The name of the standard output stream is std::cout.

How to Build the Solution

Press F7 to build the solution, or select the Build'! Build Solution in the menu. Another way is to click the toolbar button corresponding to this menu item. The Build menu toolbar buttons may not be displayed, but this can be resolved by right-clicking in the toolbar area or you can do this by selecting the Build toolbar. Then compile the program and check that there is any error. After compilation of the program, build the program successfully.

On successful compilation of the program without error, you can project folder using Windows Explorer to see a new subfolder to the solution folder Ex1_01 called Debug. It consists of output of the build solution. This folder contains three files. There is one file in executable form that is .exe file, there is no need to have knowledge about other two files. After rebuilding the project, the .ilk file is used by the linker. It allows the linker to link the object files generated from the modified source code into the existing .exe file. This avoids the requirement to re-link everything every time you edit your program. The .pdb file consists of debugging information which is used while executing the program in debug mode, where you can examine information that is produced during program execution. A Debug subdirectory also exist in the Ex1_01 project file, which consists of ten more files that were created throughout the build process.

Debug and Release Versions of Program

A range of options for a project can be seen through the Project'! Ex1_01 Properties menu item. These options regulate how the source code is managed during the compile and link stages. The set of options generates specific particular executable version of your program which is known as configuration. After creating a new project workspace, VC++ repeatedly creates configurations for generating two versions of your application. In between these versions, the one version, is known as called the Debug version. This version consists of the information that helps to debug the program and when things go wrong, you can step through the

NOTES

code as well as checking on the data values in the program. Another version is known as the Release version, which has no debug information and has the code optimization options of the compiler turned on. It delivers the most effective executable module. These two configurations are adequate. You must add another configurations for an application, by going through the Build'! Configuration Manager Menu. If, you haven't got a project loaded then this menu item won't appear. On selecting the configuration from the Active solution configuration drop-down list in the Configuration Manager Dialog box, you can select the configuration of your program to work as shown in Figure 1.14.

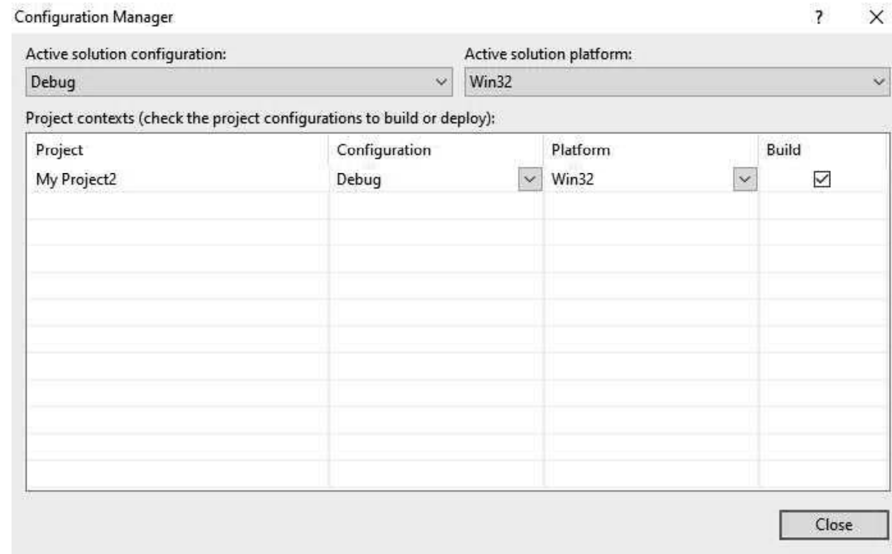


Fig. 1.14 *Configuration Manager Dialog Box*

You have to select the configuration which you want to work with from the list and click the Close button. During developing the application, you can also work with the debug configuration. After testing the application using the debug configuration and your program seems to working fine. Then, rebuild the program as a release version which generates optimized code without the debug and trace capability. The program runs faster and also occupies less memory.

Executing the Program

After successfully compilation of the solution, execute the program by pressing Ctrl+F5. Execution of the program is shown in Figure 1.15.

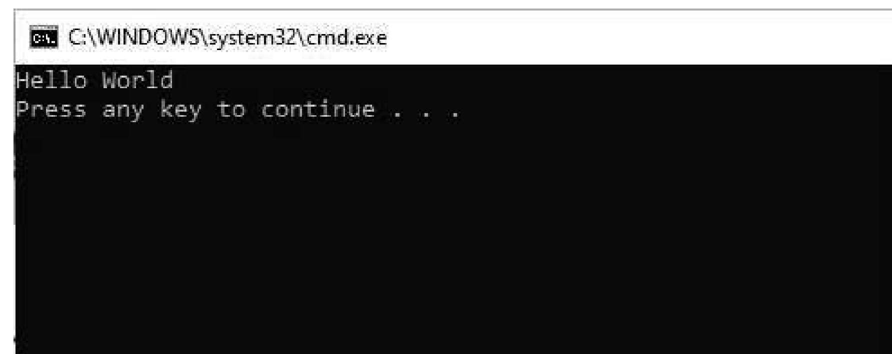


Fig. 1.15 *Executing the Program*

The text between the double quotes written to the command line is displayed in the output. The "\n" that looks at the end of the text string is known as escape sequence. This signifies a newline character. The Escape sequences are mainly used to signify characters in a text string which can't be enter directly from the keyboard.

NOTES

1.5 DIALOG BOXES

Dialog boxes are popup windows which simultaneously combine several child window controls on their surface. This multiplicity of controls allows the obtaining of a great deal of information from the user. By using a dialog box resources ID, the resource editor of Visual C++, creates a dialog box and its controls by a simple statement.

Types of Dialog Boxes

Dialog boxes are of two types, modal and modeless. Modal dialog boxes do not allow the user to switch to another window created by the same application till the user deals with the dialog box first, either by providing the requested info or by dismissing the dialog box. Modeless dialog boxes allow the users to activate other windows created by the same application, leaving the dialog box standard on the screen. They can return to the dialog box later and fill in the info being requested by it.

In MFC, both these types of dialog boxes are handled by the class CDialog. The CDialog class defines message handlers for OK and Cancel buttons.

Designing Dialog Boxes

MFC provides the Resource Editor to design dialog boxes. The format of the dialog box is created by choosing insert >> Resource or by pressing the Ctrl + R keystroke.

This will display the list of resources.

NOTES

To design the box the 'Dialog' resource type has to be selected. Visual C++ will open a dialog editor window, which displays a full-sized replica of the new dialog box. Initially, the dialog box has two controls— 'OK' and 'Cancel' push buttons.

When the dialog editor is active, the Controls and Dialog toolbars will normally be displayed. The controls toolbar contains a button for each type of control that can be added to the dialog box. The Dialog toolbar provides an alternative way to issue many of the menu commands that are used in designing a dialog box.

To add a particular type of control, an appropriate button within the control bar has to be clicked first, then the target location within the dialog box has to be clicked.

After inserting control within the dialog box, properties of the control can be set by double clicking within the control.

Creating Modal Dialog Box

The modal dialog box is first created by designing the dialog box in Resource editor. Once the dialog box has been created using the Resource Editor, the next step is to derive a class from the CDialog MFC class. The derived class defines the behaviour of a dialog box. An object of the derived class is created in the stack, then the DoModal( ) function is used to create and display the dialog box and associated controls.

```
class newdialog : public CDialog
{
 public:
newdilog ( n )
{
}
// handlers, if any would go here

};

newdilog d ( IDD_DILOG 1);
d.DoModal ( ) ;
```

Initialization of Controls Within Dialog Box (WM_INITDIALOG)

Before displaying the dialog box and after creating the controls, Windows sends the dialog box a `WM_INITDIALOG` message. This message activates the `OninitDialog()` handler. In this function, the necessary initialization is done to prepare the controls for action. On returning TRUE from `OnInitDialog()`, Windows assigns the input focus to the first control in the tab order. The user can provide his own implementation of `OnInitDialog()` function by overriding this function.

In addition to the `OnInitDialog()` function, Windows also provides two virtual functions corresponding to the OK and the Cancel buttons. These functions correspond to dialog messages. They do not need to be included in message map entries.

The `OnOK()` function is called on by clicking the OK button, `OnCancel()` function is called on by clicking the Cancel button. The `OnOK()` and `OnCancel()` functions call `EndDialog()` to dismiss the dialog box.

Check Your Progress

8. Define the terms resource view, file view and class view.
9. What is wizard bar?
10. When you use Windows Explorer then which types of files displays?
11. What are the types of dialog boxes?
12. What is the function of WM_INITDIALOG message?

1.6 CONTROLS

Controls are basically user interface objects like push buttons, list boxes, scroll bars etc. Controls are used to supply inputs. A control is actually a window, complete with its own window procedure.

An application that uses a control, does not have to draw the push button on the screen. The control's own WM_PAINT handler paints the button on the screen and the other message handler inside the control translates the user's mouse and keyboard input into notification for the control's parent.

Controls are often known as the child windows of the mainframe window. Controls are the objects of predefined MFC classes (see Table 1.1).

Table 1.1 *Control Types and their WND and MFC Classes*

Control Type	WND Class	MFC Class
Buttons	BUTTON	CButton
List boxes	LISTBOX	CListBox
Edit controls	EDIT	CEdit
Combo boxes	COMBOBOX	CComboBox
Scroll bars	SCROLLBAR	CStatic

1.6.1 Button Control

Button controls are of four types : push buttons, check boxes, radio buttons and group boxes. Push buttons are 3D rectangles that appear to go up and down when clicked. Check boxes are small squares in which check marks can be toggled on and off. Radio buttons are small circles that are checked with a solid dot when clicked. Radio buttons are usually used to present a list of mutually inclusive options to the user. A group box control considers the radio button it holds as a groups. A button can be clicked with the left mouse button or the spacebar. The spacebar works only if the button has the input focus.

MFC provides all the functionality of the standard Windows' button in the Cbutton, which is derived from the CWnd class.

NOTES

List Box

A List box is used to display a list of text strings called items. A list box item can be a string, file name, an address, a roll number, etc. A list box can have capabilities like sorting strings or scroll bars. A standard list box displays strings in a vertical column and allows only one item to be selected at a time. A list box may have multi-selected items and multiple columns.

The most common use of list boxes is in the dialog boxes. The list box typically allows a user to select a filename, directory, and so on.

Static Controls

Static controls contain the data which does not change during the execution of the programs.

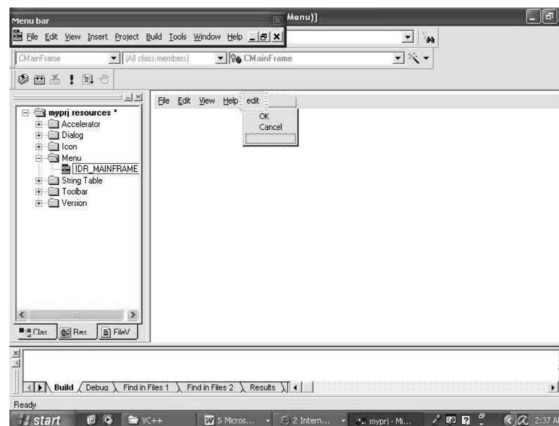
They may contain text strings and rectangles to group related controls and images formed from bitmaps, icons, cursors or metafiles.

Combo Box

A combo box is a combination of a single line edit control and a list box. Combo boxes are of three types, namely simple combo box, drop-down combo box and drop-down list box. In a simple combo box, the list box is displayed permanently. Text may also be typed into a simple combo box's edit control. A drop-down combo box is similar to a simple combo box but, a list box is displayed only when the user clicks the downward pointing arrow to the right of edit control. A drop-down list box works similarly, but does not permit the text to be typed into the edit control.

MFC provides all the functionalities of a standard Windows, combo box in the class CcomboBox, which is derived from the CWnd class.

Like the ClistBox class, the CcomboBox class provides a set of methods that make the working with the list box items relatively easy.



Edit Controls

Edit controls are used for text entry and editing. These controls have built0in support for operations like cut, copy, paste, undo, etc. They can be either single-line or multi-line.

MFC provides the services of a standard windows edit control in the class 'Cedit', which is derived from the CWnd class. An edit control can be created as a child control of any window with a code, but is typically defined in a dialog resource template.

Scroll Bar Controls

Scroll bar controls are similar to the scroll bars attached to the windows. A window scroll bar runs the full length or breadth of a window and is glued to the window border, but a scroll bar control can be placed anywhere in the window and can be of any reasonable height or width.

1.7 CREATING CONTROLS

All window controls can be created by `CWnd` member function `Create()`.

Syntax :

```
BOOL Create( DWORD dwStyle, const RECT &rect, CWnd*
pParentWnd, UINT  nId);
```

The first parameter `dwStyle` is the style of the windows control. This could be any combination of the windows' styles and the specific control styles. All controls should have WS_CHILD style. The second parameter specifies the size of the control. The third parameter `pParentWnd` is a pointer to the owner of the control. The last parameter `nID` is the control ID by the parent to communicate with the control.

Button Styles

Buttons can have a combination of windows styles, In addition to these, a button can have specific styles (Table 1.2).

NOTES

Table 1.2 *Button Style Types and their Meaning*

Style Macro	Meaning
BS_CHECKBOX	This style creates a check box (a small square with a text caption displayed to its right by default, but to its left if this style is combined with the BS_LEFTTEXT style.)
BS_3STATE	This style is just like a check box, but the box can be checked or dimmed (to show that the check box is disabled.)
BS_AUTO3STATE	This style is just like a checkbox, but the box can be checked or dimmed(to show that a check box is disabled). When the user selects the box, the check state of the button is toggled automatically.
BS_AUTOCHECKBOX	This style is just like the checkbox but when the user selects the checkbox, the checked state of the button toggles automatically.
BS_AUTORADIOBUTTON	This style is just like a checkbox, when user selects the box, the checked state of the button toggles automatically.
BS)_DEFPUSHBUTTON	This style creates a default push button(a button with a heavy black border) that allows the user to quickly select the default command option by pressing the Enter key.
BS_DISABLENOSCROLL	This sytle creates a captioned framing rectangle for visually grouping controls.
BS_LEFTTEXT	When combined with the radio button or the checkbox style, this style causes the button text to be displayed on the left side of the radio button or checkbox.
BS_OWNERDRAW	This style creates an owner-drawn button. MFC automatically calls the `Drawitem()` method when the button changes visually. This style must be set when using the `CbitmapButton` class.
BS_PUSHBUTTON	This style creates a push button (a button thay sends a WM_COMMAND message to the owner window when the user clicks the button.
BS_RADIOBUTTON	This style creates a radiobutton (a small circle with a text caption displayed to its right by default, but to its left if this style is combined with a related but mutually exclusive choice.

Cbutton messages

Since MFC wraps the standard Window control messages into Cbutton class methods, an MFC message amp usually has to handle only the button notification messages.

A button sends notification messages to its owner window (see Table 1.3).

Table 1.3 *Meaning of the Various Message Map Entries*

Message Map Entry	Meaning
ON_BN_CLICKED	Sent by a button control when a user clicks the button.
ON_BN_DBCLICKED	Sent by a button control when a user double-clickes the button.
ON_COMMAND	Sent by a button control when a user clicks the button (same as ON_BN_CLICKED)

NOTES

Example

```
class CMainFrame : public CFrameWnd
{
private:
    CButton button;
public:
    CMainFrame();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL DestroyWindow();
    virtual CDocument* GetActiveDocument();
    //}}AFX_VIRTUAL
// Implementation
public:
    virtual ~CMainFrame();

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)
```

NOTES

```
// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnPaint();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

Define Oncreate() message handler as in frame window
class as:

int mainframe : OnCreate(LPCREATESTRUCT)
{
 CFRAMEWND ::OnCreate();
Button_Create("OK", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE,
CRect(200,100,300,250); this ,1);
Button,Setfocus();
Return 0 ;
|
```

Message map contains notification of events on button as :

```
BEGIN_MESSAGE_MAP(MyFrame, CFrameWnd)
                   ON_WM_CREATE()
                    ON_COMMAND(1,OK)
END_MESSAGE_MAP()
```

1.7.1 List Box

A list box can take all the standard Windows styles as well as some specific styles (Table 1.4).

Table 1.4 *List Box Style Macros and their Meaning*

Style Macros	Meaning
LBS_MULTIPLESEL	Allows multiple strings to be selected from the same list box. Selection for a particular string can be turned ON or OFF by clicking or double-clicking the string or pressing the spacebar.
LBS_EXTENDEDSEL	Allows quick selection in multiple-selection boxes using the shift key and the mouse or through special key combinations.
LBS_MULTICOLUMN	Creates a special type of list box, which has several horizontally scrollable columns. The LB_SETCOLUMN WIDTH message can be used to set the width of these columns.
LBS_NOTIFY	Turns on the WM_COMMAND messages which notify the parent window whenever the user selects or deselects any string.
LBS_SORT	Forces the listbox to sort all the incoming strings alphabetically as it adds them to the list. This means that strings are arranged in a different order than the one in which they were sent.
LBS_STABDARD	Combines the LBS_SORT, LBS_NOTIFU, WS_VSCROLL and WS_BORDER styles/
LBS_DISABLENOSCRO LL	Prevents the vertical scrollbar from being hidden when the number of items in the listbox becomes too small to allow vertical scrolling. This causes a disabled vertical scrollbar to appear all the times, regardless of the number of items. If there are enough items, the scrollbar is enabled.
LBS_HASSTRINGS	Forces the listbox to allocate memory and make copies of all strings that are added to it. The listbox automatically manages all the pointers and memory operations required for this purpose.
LBS_NOINTEGRAL HEIGHT	When they arecreated, list boxes usually adjust their height in such a way that they become a multiple of character height and the no partial strings are shown. This style disables this height readjustment and final height of the listbox is the same as specified in the `CreateWindow()` call.
LBS_NOREDRAW	Prevents the listbox from redrawing itself whenever a new string is added to it. It speeds up the initialization of the list box by avoiding repeated repainting every time a string gets added. The WM_SERREDRAW message can be used to turn on the redrawing, after the list box has been created.
LBS_USETABSTOPS	Forces a list box to expend tab characters when drawing its strings. The default tab positions are 32 dialog box units, i.e. 8 characters wide. The LBS_SETTABSSTOPS message can be used to change this to any other value.

NOTES

CListBox messages

MFC wraps the standard Window list box message into a CListBox class method. An MFC program usually needs to handle only the notification messages. The list box sends notification messages to its owner window. These messages can be trapped and handled by writing Message Map entries and message handler methods for each message (Table 1.5).

NOTES

Table 1.5 *Message Map Entries and and their Meaning*

Message Map Entry	Meaning
ON_LBN_DBLCLK	List boxes with the LBS_NOTIFY style, send this message to the owner when a user double clicks an item in a list box.
ON_LBN_ERRSPACE	The list box can not allocate enough memory to meet the request.
ON_LBN_KILLFOCUS	This message occurs when a list box looses the input focus.
ON_LBN_SELCANCEL	List boxes with the LBS_NOTIFY style send this message to the owner when the current list box selection has been cancelled.
ON_LBN_SELCHANGE	List boxes that have the LBS_NOTIFY style send this notification to the parent window when the selection in the list box changes. If the selection is changed by the CListBox ::`SetCurSel()` class method, the notification is not sent. For multiple selection list boxes, this notification is sent when a user presses an arrow key, even if the selection does not change.
ON_LBN_SETFOCUS	This message occurs when a list box receives the input focus .

Example

```
class CMainFrame : public CMDIFrameWnd
{.
private:
    CListBox list1;
    CButton b1,b2;
    DECLARE_DYNAMIC(CMainFrame)
public:
    CMainFrame();
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL DestroyWindow();
    virtual CDocument* GetActiveDocument();
    //}}AFX_VIRTUAL
// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;
// Generated message map functions
protected:
```

NOTES

```
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void Additem();
    afx_msg void DeleteItem();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_COMMAND(Additem, Additem)
    ON_COMMAND(delitem, DeleteItem)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
Message Handler function is written as:
int Mainframe :: OnCreate(LCOREATESTRUCT 1)
{
    CFamreWnd  :: OnCreate( 1 );
    List1.CreateEx( WS_EX_CLIENTEDGE, " LISTBOX", NULL,
WS_CHILD | WS_VISIBLE | WS_BORDER | WS_VSCROLL |
LBS_DISABLENOSCROLL. Crect( 100, 200,300, 350 ), this , 1
);
List1.AddString("ADCA");;
List1.AddString("DCA");;
List1.AddString("PGDCA");;
b1.Create("Add, - , -, -, -);
b2.Create("Delete", -, -, -, - );
return 1;
}
void CMainFrame ::Additem()
{
list1.AddString( "M.Tech  LEVEL");
list1.AddString( "B.Tech LEVEL");
list1.AddString( "MCA LEVEL");
list1.AddString( "PGDCA LEVEL");
b1.EnableWindow(FALSE);
}
void CMainframe :: Deleteitem()
{
int index;
    Index = list1.FindStringExact(-1 , "M.Tech Level);
    list1.DeteleString(index);
index = list1.FindStringExact(-1 , "MCA Level);
    list1.DeteleString(index);
index = list1.FindStringExact(-1 , "PGDCA Level);
    list1.DeteleString(index);
b2.EnableWindow(FALSE);
}
```

NOTES

ClistBox Methods

A list box contains three basic methods.

`AddString()` : It is used to add a string to the list box.

`DeleteString()` : It is used to delete a string from the list box.

`FindStringExact()` : It is used to search from the first string in the list box that matches a specified search string.

Edit Control Styles

Like all windows, edit control can use the style available to `CWnd`, In addition edit control can have specific styles (Table 1.6).

Table 1.6 *Edit Style Macros and their Meaning*

Style MACROS	Meaning
ES_AUTOHSCROLL	Automatically scrolls the text to the right by 10 characters, when a user types a character at the end of the line. When a user presses the enter key, the control scrolls all text back to position zero.
ES_AUTOVSCROLL	Automatically scrolls text up one page when a user presses the enter key on the last line.
ES_CENTER	Centers text in a multiple line control.
ES_LEFT	All text is left justified.
ES_LOWERCASE	Automatically converts all the characters to lower cases. They are typed into the edit control.
ES_MULTILINE	Specifies that an edit control is multiple line edit control (instead of the default single line control).
ES_NOHIDESEL	Supresses the default action of an edit control, which is to hide selected text when the control looses the input focus and to invert the selection when the control receives the input focus.
ES_NUMBER	Windows 95 only .allows digits to be entered into the edit control.
ES_OEMCONVERT	Ensures proper character conversion when the application calls the windows, API function `AnsiToOem()` to convert an ANSI string in the edit control to OEM character `O` and item sets it back to ANSI. This style is most useful for edit controls that contain Filenames.
ES_PASSWORD	Displays all the characters as asterisks (*) when they are typed into the edit control. An application can use the `SetPasswordChar()` method to display a different character instead.
ES_READONLY	Prevents a user from entering or editing the text in the edit control.
ES_RIGHT	All text is right justified in a multiple-line edit control.
ES_UPPERCASE	Converts all the characters to uppercase when they are typed into the edit control.

NOTES

CEdit messages

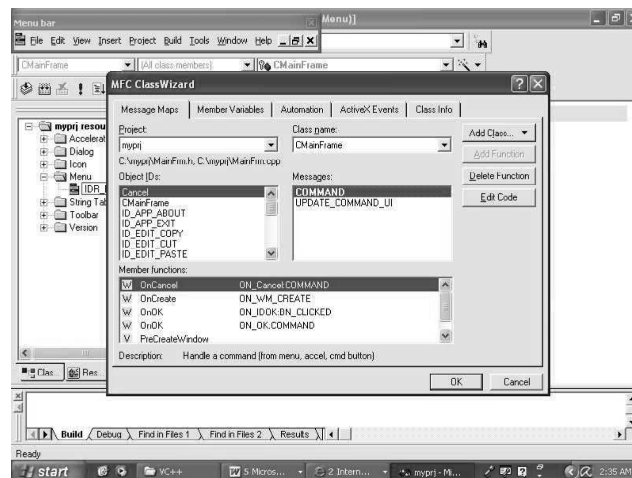
MFC wraps the standard Windows, edit messages into CEdit class methods; an MFC program usually handles only the notification messages. An edit control sends notification messages to its owner, which is usually CDialog derived class. These messages can be trapped and handled by writing message map entries and message handler methods for each of the messages. The message map entries and methods are implemented in the edit control's owner class (see Table 1.7).

***Table 1.7** Message Map Entries for CEdit Messages*

Message Map Entry	Meaning
ON_EN_CHANGE	Sent when a user has changed text in an edit control. Unlike the EN_UPDATE notification message, this notification message is sent after Windows updates the display.
ON_EN_ERRASPACE	Sent when an edit control can't allocate enough memory to meet a specific text request.
ON_EN_HSCROLL	Sent when a user clicks an edit control's horizontal scroll bar. The parent window is notified before the screen is updated.
ON_EN_KILLFOCUS	Sent when an edit control loses the input focus.
ON_EN_MAXTEXT	Sent when the number of characters in an edit control has exceeded the specified number (or maximum allowed number) of characters for the edit control and the text has been truncated. This notification is also sent when an edit control doesn't have the ES AUTOHSCROLL style and the number of characters would exceed the width of the edit control, or when an edit control doesn't have the ES_AUTOVSCROLL style and the total number of lines resulting from a text insertion would exceed the height of the edit control.
ON_EN_SETFOCUS	Sent when an edit control receives the input focus.
ON_EN_UPDATE	Sent after an edit control's altered text has been reformatted, but before it is displayed.
ON_EN_VSCROLL	Sent when a user clicks on the edit control's vertical scroll bar. The parent window is notified before the screen is updated.

Example: Creating a single and multi-line Edit control.

- Create menu items OK and Cancel in the main menu of the frame window with the IDs OK and Cancel, respectively.
- Add message handler functions corresponding to these menu items.



NOTES

```
class CMainFrame : public CFrameWnd
{

private :
    CEdit ed1 , ed2 ;
    CButton b[3];
    CStatic str1 , str2;
protected: // create from serialization only

    CMainFrame();
int OnCreate(LPCREATESTRUCT!);

    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnOK();
    afx_msg void OnCancel();
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
};

// MainFrm.cpp : implementation of the CMainFrame class
//
#include "stdafx.h"
#include "myprj.h"
#include "MainFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////
/////////////////////
// CMainFrame
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_COMMAND(OK, OnOK)
    ON_COMMAND(Cancel, OnCancel)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////
/////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here

}
```

NOTES

NOTES

```
CMainFrame::~CMainFrame()
{
}
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
       return -1;
    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD
| WS_VISIBLE | CBRS_TOP
       | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_SIZE_DYNAMIC) ||
       !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
       TRACE0("Failed to create toolbar\n");
       return -1;      // fail to create
    }
    if (!m_wndStatusBar.Create(this) ||
       !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
    {
       TRACE0("Failed to create status bar\n");
       return -1;      // fail to create
    }
    // TODO: Delete these three lines if you don't want
the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);
ed1.Create("Name", WS_CHILD | ES_AUTOHSCROLL | WS_VISIBLE
, CRect( 100, 20 , 160, 45 ) , this , 1 );
ed2.Create("Address", WS_CHILD | ES_AUTOHSCROLL |
WS_VISIBLE , CRect( 200, 40 , 300, 45 ) , this , 2 );
str1.Create("Name", WS_CHILD } WS_VISIBLE | SS_RIGHT ,
CRect(-,-,-,-), this 3);
str2.Create("Address", WS_CHILD } WS_VISIBLE | SS_RIGHT
, CRect(-,-,-,-), this 3);
b[0].Create("OK", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE,
CRect(-,-,-,-) , this , 4);
b[1].Create("Cancel", BS_PUSHBUTTON | WS_CHILD |
WS_VISIBLE, CRect(-,-,-,-) , this , 4);
ed3.Create(WS_EX_CLIENTEDGE, "EDIT", " ",WS_CHILD |
ES_MULTILINE | ES_READONLY | WS_VISIBLE , CRect( 200, 40
, 300, 45 ) , this , 7 );
    return 0;
}
```

NOTES

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
       return FALSE;
    // TODO: Modify the Window class or styles here by
modifying
    //  the CREATESTRUCT cs
    return TRUE;
}
/////////////////////////////////////////////////////////
/////////////////////
// CMainFrame diagnostics
#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}
#endif //_DEBUG
/////////////////////////////////////////////////////////
/////////////////////
// CMainFrame message handlers
void CMainFrame::OnOK()
{
    // TODO: Add your command handler code here
    CString str, temp;
    ed1.GetWindowText(temp);
    str = "Name" + temp;
    ed2.GetWindowText(temp);
    str = str + "\n\nAddress :\t" + temp;
    ed3.SetWindowText(str);
}
void CMainFrame::OnCancel()
{
    // TODO: Add your command handler code here
    ed1.SetWindowText(" ");
    ed2.SetWindowText( " ");
}
```

NOTES

1.7.2 Combo Box

Like all windows, combo boxes can have a combination of window styles as well as specific styles (Table 1.8).

Table 1.8 *Combo Box Style Macrons and their Meaning*

Style Macro	Meaning
CBS_AUTOHSCROLL	Automatically scrolls the text in the combo box's child edit control to the right when a user types a character at the end of the line. If this style is not set, only the text that fits within the rectangular boundary is allowed.
CBS_DIABLENSCROLL	Disables (grays) bar instead of making it disappear when a combo box doesn't contain enough items to require scrolling.
CBS-DROPDOWN	The list box portion isn't displayed unless a user selects the drop-down portion next to the edit control; the current selection in the list box portion is displayed in the edit control.
CBS_DROPDOWNLIST	The list box portion isn't displayed unless a user selects the drop-down button next to a static text item (that replaces the edit control), which displays the current selection in the list box.
CBS_HASSTRINGS	Specifies an owner-drawn combo box that contains string items. The list box portion takes care of allocating memory for the strings and text for a specific item can be retrieved with the `Get that Text( )` methods.
CBS_LOWERCASE	Converts all the text to lowercase in both the selection field and the list.
CBS_NOINTEGRALHEI GHT	Windows ordinarily sizes a combo box, so that it doesn't display partial items if the box isn't big enough. This macro sizes the combo box exactly as specified at creation, showing partial items if they are present.
CBS_OEMCONVERT	Ensures proper character conversion when the application calls the Windows API function `AnsiToOem( )` to convert an ANSI string in the edit control to OEM characters by converting characters entered in the control from the ANSI character set to the OEM character set and then back to ANSI. This style is most useful for combo boxes that contain filenames and applies only to the combo boxes created with the CBS_SIMPLE or CBS_DROPDOWN styles.
CBS_SIMPLE	The list box portion is always visible and the current selection in the list box is displayed in the combo box's child edit control.
CBS_SORT	Automatically sorts list box strings in a combo box.
CBS_UPPERCASE	Converts all text to uppercase in both the selection field and the list.

CcomboBox messages

MFC wraps the standard windows combo box messages into CcomboBox class methods. An MFC program usually has to handle only the notification messages.

A combo box sends the notification messages to its owner. These messages can be trapped and handled by writing message map entries and message handler

methods for each message. The message map entries and methods are (Table 1.9) implemented in the combo box's owner class.

Table 1.9 *Message Map Entries and their Meaning*

NOTES

Message Map Entry	Meaning
ON_CBN_CLOSEUP	Sent when the list box of a combo box (except those with the CBS_SIMPLE style) has closed.
ON_CBN_DBLCLK	Sent when the user double-clicks a list item in a combo box with the CBS_SIMPLE style. Note that double-clicks can't occur for combo boxes with the CBS_DROPDOWNLIST style because a single click hides the list box.
ON_CBN_DROPDOWN	Sent when the list box of a combo box (CBS_DROPDOWN or CBS_DROPDOWNLIST style) is about to drop, exposing the list items.
ON_CBN_EDITCHANGE	Sent when a user has altered the text in the edit control of a combo box. This message is sent after Windows updates the screen (unlike the CBN_EDITUPDATE message). This message isn't sent at all for drop-down list combo boxes (those with the CBS_DROPDOWNLIST style).
ON_CBN_EDITUPDATE	Sent just before the edit control of a combo box displays altered text, but after the control has formatted the text. This message isn't sent at all for drop-down list combo boxes (those with the CBS_DROPDOWNLIST style).
ON_CBN_ERRSPACE	Sent if a combo box can't allocate enough memory to meet a specific request.
ON_CBN_KILLFOCUS	Sent when the combo box loses the input focus.
ON_CBN_SELCHANGE	Sent when a user either clicks in the list box or changes the selection with the arrow keys on the keyboard.
ON_CBN_SELENDCHANGE	Sent in response to a user clicking a combo box list item and then clicking away from the list to another window or control, hiding the list box portion. This message is sent before the CBN_CLOSEUP notification message to indicate that user's selection should be ignored.
ON_CBN_SELENDOK	Sent when a user selects an item and then closes the list. This notification message is sent before the CBN_CLOSEUP message to indicate that the user's selection should be considered valid.
ON_CBN_SETFOCUS	The combo box receives the input focus.

CcomboBox methods

`GetCurSel():`

It gets the index of the currently selected item, if any, one in the list box of combo box.

```
Syntex  : int GetCurSel();
```

`SetCurSel():`

It selects a string in the list box of a combo box.

```
Syntex  : void SetCurSel(int);
```

NOTES

AddString():

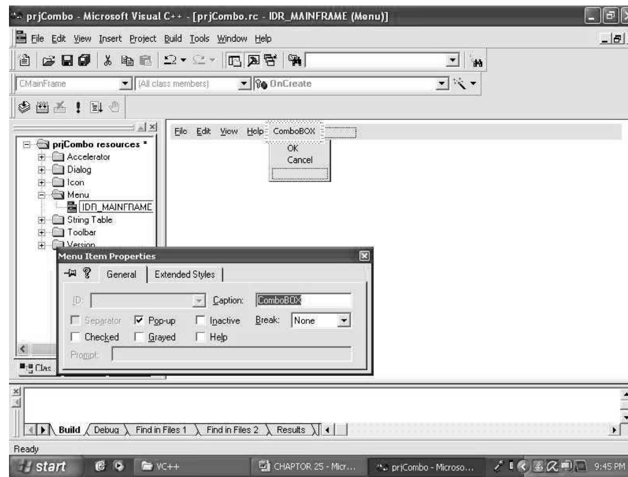
It adds a string to the end of the list in the list box portion of a combo box or at the sorted position for list boxes with the CBS_SORT Style.

```
Syntex  : void AddString(Cstring);
    Example : Creating Combobox in a Window.
    class CMainFrame : public CFrameWnd
    {
    private :
       CComboBox cb;
       CButton b[2];
    protected: // create from serialization only
       CMainFrame();
       DECLARE_DYNCREATE(CMainFrame)

    // Attributes
    public:
    // Operations
    public:
    // Overrides
       // ClassWizard generated virtual function overrides
       //{{AFX_VIRTUAL(CMainFrame)
       virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
       //}}AFX_VIRTUAL
    // Implementation
    public:
       virtual ~CMainFrame();
    #ifdef _DEBUG
       virtual void AssertValid() const;
       virtual void Dump(CDumpContext& dc) const;
    #endif

    protected:  // control bar embedded members
       CStatusBar  m_wndStatusBar;
       CToolBar   m_wndToolBar;
    // Generated message map functions
    protected:
       //{{AFX_MSG(CMainFrame)
       afx_msg   int   OnCreate(LPCREATESTRUCT
    lpCreateStruct);
          // NOTE - the ClassWizard will add and remove
    member functions here.
          //    DO NOT EDIT what you see in these blocks
    of generated code!
       //}}AFX_MSG
       DECLARE_MESSAGE_MAP()
```

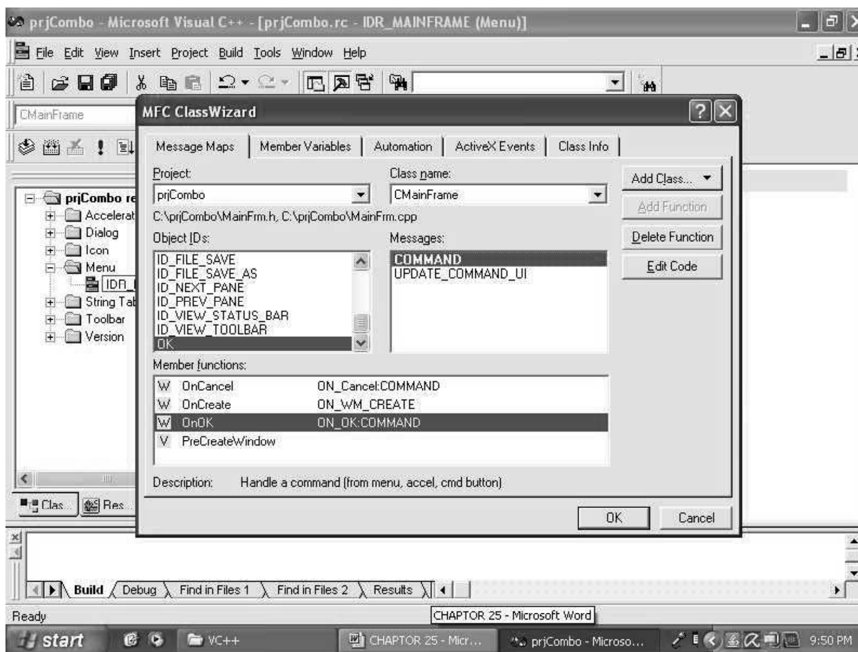
```
};
////////////////////////////////////////////////////
/////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional
declarations immediately before the previous line.
#endif                                          //
!defined(AFX_MAINFRM_H__7CB5E539_3339_4424_8FF5_C486
FF5B2E20__INCLUDED_)
```

NOTES

- Create a menu item ComboBox with two menu items, OK and Cancel, with ID OK and Cancel respectively, to handle the actions on the combo box.



- Add message handler function corresponding to WM_CREATE event, to create the combo box and the message handler function to perform on the combo box on WM_COMMAND message of the two menu items OK and Cancel, using ClassWizard.



```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "prjCombo.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////////////////////////////////////////////////
/////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_COMMAND(Cancel, OnCancel)
    ON_COMMAND(OK, OnOK)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

///////////////////////////////////////////////////////
/////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here
    CString mywindowclass;
    mywincls = AfxRegisterWndClass( CS_HREDRAW | CS_VREDRAW
```

NOTES

```
, 0 , ( HBRUSH ) :: GetStockObject( LTGRAY_BRUSH) , 0 );
    Create(myWinCls , "ComboBox");

}
CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{



    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
       return -1;

    cb.CreateEx( WS_EX_CLIENTEDGE | "COMBOBOX" , ** ,
WS_CHILD | WS_VISIBLE | CBS_DropDownLIST | CBS _SORT |
WS_VSCROLL , CRect(-, -, -, -, - ), this , 1);
    cb.AddString ( "ID!_ICON1 ");
    cb.AddString ( "ID!_ICON2 ");
    cb.AddString ( "ID!_ICON3 ");
    cb.AddString ( "ID!_ICON4 ");
    cb.AddString ( "ID!_ICON5 ");
    cb.SetCurSel( 0 ) ;
    b[0] . Create ( "OK" , BS_PUSHBUTTON | WS_CHILD |
WS_VISIBLE | , CRect (- ,- ,-, -) , this , 1);
    b[1]. Create( "Cancel" , BS_PUSHBUTTON | WS_CHILD |
Ws_VISIBLE | , CRect (-, -, -,- ) , this , 1);

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD
| WS_VISIBLE | CBRS_TOP
       | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_SIZE_DYNAMIC) ||
       !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
       TRACE0("Failed to create toolbar\n");
       return -1;      // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
       !m_wndStatusBar.SetIndicators(indicators,
         sizeof(indicators)/sizeof(UINT)))
    {
       TRACE0("Failed to create status bar\n");
       return -1;      // fail to create
```

NOTES

NOTES

```
    }
    // TODO: Delete these three lines if you don't want
the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
       return FALSE;
    // TODO: Modify the Window class or styles here by
modifying
    //  the CREATESTRUCT cs

    return TRUE;
}

//////////////////////////////////////////////////////
////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

//////////////////////////////////////////////////////
////////////////////
// CMainFrame message handlers

void CMainFrame::OnCancel()
{
```

```
    // TODO: Add your command handler code here
    MessageBox ( " Selection is Cancelled " , "Cancel");

}

void CMainFrame::OnOK()
{
    // TODO: Add your command handler code here
    switch( cb.GetCurSel() )
    {
    case 0 :
       MessageBox("ID!_ICON1" , "Icon Is " );
       break;
    case 1 :
       MessageBox("ID!_ICON2" , "Icon Is " );
       break;
    case 2 :
       MessageBox("ID!_ICON3" , "Icon Is " );
       break;
    case 3 :
       MessageBox("ID!_ICON4" , "Icon Is " );
       break;
    case 4 :
       MessageBox("ID!_ICON5" , "Icon Is " );
       break;
    }

}

Resource.h file is :
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by PRJCOMBO.RC
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME             128
#define IDR_PRJCOMTYPE                  129

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                1
#define _APS_NEXT_RESOURCE_VALUE  130
```

NOTES

NOTES

```
#define _APS_NEXT_CONTROL_VALUE        1000
#define _APS_NEXT_SYMED_VALUE          101
#define _APS_NEXT_COMMAND_VALUE        32771
#endif
#endif
```

Application Class :

```
// prjCombo.h : main header file for the PRJCOMBO
application
//

#if !defined(AFX_PRJCOMBO_H__4495F482_4AD2_4901_B7FC
_10542BC8291E__INCLUDED_)
#       d       e       f       i       n       e
AFX_PRJCOMBO_H__4495F482_4AD2_4901_B7FC_10542BC8291E
__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file
for PCH
#endif

#include "resource.h"      // main symbols

////////////////////////////////////////////////////
////////////////////
// CPrjComboApp:
// See prjCombo.cpp for the implementation of this class
//

class CPrjComboApp : public CWinApp
{
public:
    CPrjComboApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPrjComboApp)
    public:
    virtual BOOL InitInstance();
```

NOTES

```
    //}}AFX_VIRTUAL

// Implementation
    //{{AFX_MSG(CPrjComboApp)
    afx_msg void OnAppAbout();
        // NOTE - the ClassWizard will add and remove
member functions here.
        //    DO NOT EDIT what you see in these blocks of
generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////
/////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif                                                  //
!defined(AFX_PRJCOMBO_H__4495F482_4AD2_4901_B7FC_10542BC8291E__INCLUDED_)
```

Application source file is :

```
// prjCombo.cpp : Defines the class behaviors for the
application.
//

#include "stdafx.h"
#include "prjCombo.h"

#include "MainFrm.h"
#include "prjComboDoc.h"
#include "prjComboView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////
/////////////////////
// CPrjComboApp
```

NOTES

```
BEGIN_MESSAGE_MAP(CPrjComboApp, CWinApp)
    //{{AFX_MSG_MAP(CPrjComboApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove
mapping macros here.
        //    DO NOT EDIT what you see in these blocks of
generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP,
CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

////////////////////////////////////////////////////////
////////////////////
// CPrjComboApp construction

CPrjComboApp::CPrjComboApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

////////////////////////////////////////////////////////
////////////////////
// The one and only CPrjComboApp object

CPrjComboApp theApp;

////////////////////////////////////////////////////////
////////////////////
// CPrjComboApp initialization

BOOL CPrjComboApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to
reduce the size
    //  of your final executable, you should remove from
the following
```

NOTES

```
    //  the specific initialization routines you do not
need.

#ifdef _AFXDLL
    Enable3dControls();              // Call this when
using MFC in a shared DLL
#else
    Enable3dControlsStatic();    // Call this when
linking to MFC statically
#endif

    // Change the registry key under which our settings
are stored.
    // TODO: You should modify this string to be something
appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated
Applications"));

    LoadStdProfileSettings();  // Load standard INI file
options (including MRU)

    // Register the application's document templates.
Document templates
    //  serve as the connection between documents, frame
windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CPrjComboDoc),
        RUNTIME_CLASS(CMainFrame),       // main SDI frame
window
        RUNTIME_CLASS(CPrjComboView));
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands,
DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so
show and update it.
```

```
    m_pMainWnd->ShowWindow(SW_SHOW);
```

NOTES

```
    CMainframe *p;
    p = new CMainframe;
    p -> ShowWindow ( 3) ;
    m_pMainWnd->UpdateWindow();

    return TRUE;
}

/////////////////////////////////////////////////////////
/////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    /
/ DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
       // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}
```

```
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
       // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CPrjComboApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

///////////////////////////////////////////////////////
/////////////////////
// CPrjComboApp message handlers
```

NOTES

1.7.3 Slider Control

A **Slider Control** is also known as trackbar. It is a Window containing a slider and optional tick marks. This control sends notification messages to represent the change when the user moves the slider, using the mouse or the direction keys. Horizontal and vertical are the two types of slider that is represented by **CSliderCtrl class**.

Check Your Progress

13. What are controls?
14. List the types of Button controls.
15. What does a standard list box display?
16. What are static controls?
17. What are the types of combo box?
18. Name the operations for which edit control has built in support.
19. Where can a scroll bar control be placed?
20. Name the parameters of windows controls.
21. What is the function of LBS-SORT?
22. Where does the list box send the notification messages?

1.8 MESSAGES

Events in a Windows application are signal or actions received like the user press the key or click the mouse or timer reaching zero. The Windows operating system records every event in a message and places the message in the program for which the message is proposed. So, a Windows message is record of the data which is related to the event. The message queue for an application is a sequence of messages waiting to be managed by the application.

By sending a message, Windows can tell your program that something required or some information has become available or an event occurs. If the program is well organized, then the response will be correct and appropriate to the message. Consider an example, a Windows program must have a function definitely for handling these messages. The function mostly called WindowProc() or WndProc(). Instead, it doesn't have to have a specific name because Windows accesses the function with the help of a pointer to a function which has been supplied. So, the sending of a message to your program boils down to Windows calling a function that you provide, typically called WindowProc(), and passing any important data to the program by means of arguments to this function. In the WindowProc () function, to find out what message is from the data supplied and what you have to do about it. There is no requirement to write code to process every message, instead you can select out some of those which are of interest in the program. Use only those messages in whatever way you want, and deliver back the rest one to the Windows. By calling a standard function, pass a message back to Windows, which is provided by Windows called DefWindowProc().

1.8.1 Message Queues

The WinMain() is producing with the messages that Windows may have queued for the application. The function WindowProc() is required to handle messages. There are two types of Message Queues: Queued and Non-Queued Messages.

Queued and Non-Queued Messages

The queued messages are placed in the queue that are extracted using WinMain() function for processing. The WinMain() code that does this is known as the message loop. The queued messages includes the messages which are arising from user input from the keyboard, using mouse or by clicking the mouse buttons. The Windows messages or the messages from a timer are used to request that a Window be redrawn are also queued.

The non-queued messages are processed using the WindowProc() function. These are being called directly by Windows. A large number of the non-queued messages arise as a result of processing queued messages. What has been done in the message loop in WinMain() is recovering a message that Windows has queued for your application and then asking Windows to invoke your WindowProc() function to process it.

The Message Loop

Recovering messages from the message queue is done with the help of a typical method in Windows programming known as message pump or message loop. The code for message loop is given below:

```
MSG msg; // Windows message structure
while(GetMessage(&msg, 0, 0, 0) == TRUE) // Get any messages
{
TranslateMessage(&msg); // Translate the message
DispatchMessage(&msg); // Dispatch the message
}
```

This code consists of mainly three functions to deal with each message:

1. GetMessage(): to retrieves a message from the queue.
2. TranslateMessage(): to performs any conversion required on the message retrieved.
3. DispatchMessage(): that causes Windows to call the WindowProc() function in the application for processing the message.

The GetMessage() function is essential because it has an important contribution to the way Windows works with various applications. The GetMessage() function recovers a message queued for the application window and stores information related to the message in the variable *msg.* The variable *msg* that is a struct of type MSG that consists of a number of dissimilar members. The code of the structure is given below:

```
struct MSG
{
HWND hwnd; // Handle for the relevant window
UINT message; // The message ID
WPARAM wParam; // Message parameter (32-bits)
LPARAM lParam; // Message parameter (32-bits)
DWORD time; // The time when the message was queued
POINT pt; // The mouse position
};
```

1.8.2 Handling Messages with Class Wizard

Class Wizard is used to create new MFC classes, add messages and message handlers to existing classes in the project.

There are three ways to open the Class Wizard:

1. Go to the Project menu and select the Class Wizard.
2. Then press Ctrl $\rightarrow$ Shift $\rightarrow$ X.
3. In Class View, right click on the project node or a class and select Class Wizard.

NOTES

Figure 1.16 represents the MFC Class Wizard.

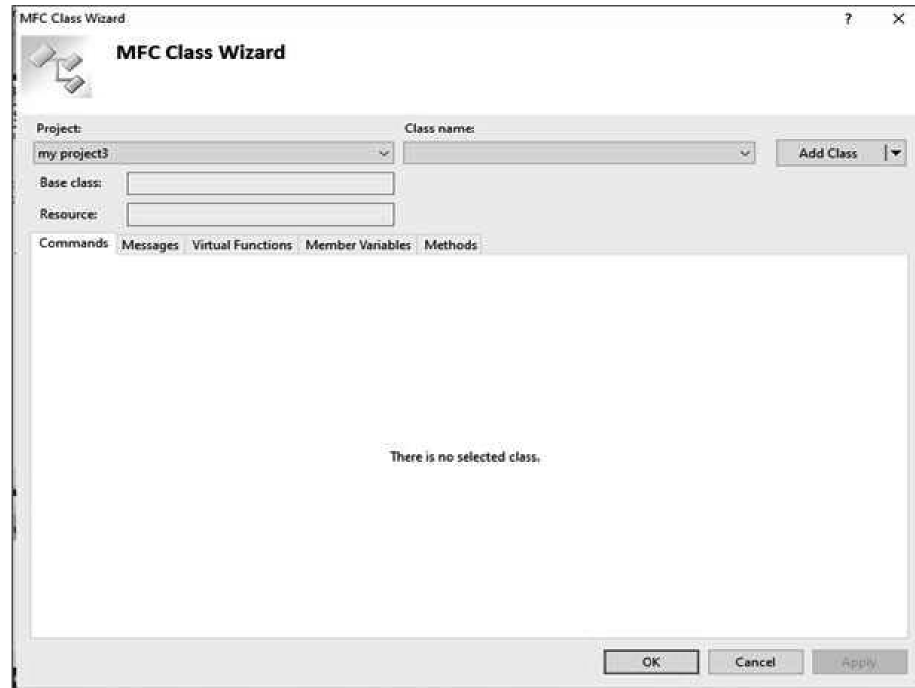


Fig. 1.16 *MFC Class Wizard*

UI Element List

- **Project:**
  - o Mention the name of a project in the solution.
  - o From the drop-down list box, you can select other projects in your solution.
- **Class declaration:**
  - o The class in which the Class name is declared.
  - o The Class declaration box is presented only if the name in it varies from the name in Class implementation.
- **Class name:**
  - o Mention the name of a class in your project.
  - o After the selection of a class in the Class name list, the data from the class occupies the controls in the MFC Class Wizard.
  - o After changing the value of a control, data in the selected class is affected.
- **Add Class:**
  - o This feature enables you add a new class to your MFC project.
- **Class implementation:**
  - o The name of the file that consists of the execution of the class in Class name.
- **Base class:**
  - o The base class of the class that is displayed in Class name.

- **Resource:**
  - o The ID of the resource in Class name, instead, the Resource box is empty.

Using the table given below, you can select a different execution file by clicking the arrow.

Option	Description
To Open File	Exits the class wizard and opens the current class execution file.
To Copy Full Path to Clipboard	It copies the path of the current execution file to the Clipboard.
Open Containing Folder	It opens the folder that consists of the current class execution file.

- **Messages:**
  - o It lets you to add, delete, search or edit for a message and its message handler.
  - o Click Add Handler to add a handler or double-click an item in the Messages list.
  - o To add a custom message, click Add Custom Message.
  - o You can Add Handler by pressing the Enter key and identify values in the Add Custom Message dialog box.
  - o In the dialog box, you can select Registered Message also to handle a window message that is guaranteed to be unique during the operating system.
- **Virtual Functions:**
  - o The Virtual Functions lets you to add, edit, search or delete for a virtual function,
  - o It also lets you to overridden virtual function.
- **Commands:**
  - o Lets you to add, delete, search or edit for a command and its message handler.
  - o Click Add Handler to add a handler, or double-click an item in the Object IDs list or Messages list.
  - o The resulting function ID, name and message are displayed in the Member functions list.
  - o To delete a handler, you have to select an item in the Member functions list and then click Delete Handler.
  - o If you want to modify a handler, then double-click the corresponding item in the Member functions list.
  - o You can modify a handler by selecting an item in the list box and then click Edit Code.
- **Methods:**
  - o Lets you to add, search or delete for a method.

NOTES

- o It also go to the definition or declaration of a method.
- o Click Add Method to add a method and then identify values in the Add Method dialog box.
- o To delete a method, you have to select an item in the Methods list and click Delete Method.
- o To display a declaration, select an item in the Methods list and then click Go to Declaration.
- o Double-click an item in the Methods list, to display a definition.
- o You can display a definition by selecting an item in the Methods list and then click the Go to Definition button.

- **Member Variables:**
  - o Lets you add, edit, search or delete for a member variable.

1.9 DOCUMENT AND VIEWS

Multiple Document Interface (MDI) is a standard way to write an application in which one master window holds a number of child windows (Figure 1.17). MDI is a user interface specification that allows the user to work with many documents at a time in a single application. In an MDI application, each document is displayed in a separate child view window. Each view child window is displayed on the client area of the main application window, which is the parent of all the child windows. The child view object in an MDI application is created inside the childframe object, which itself is contained in the application's mainframe object.

The most popular Windows programs that use this interface are Microsoft Excel and word processing softwares showing many documents.

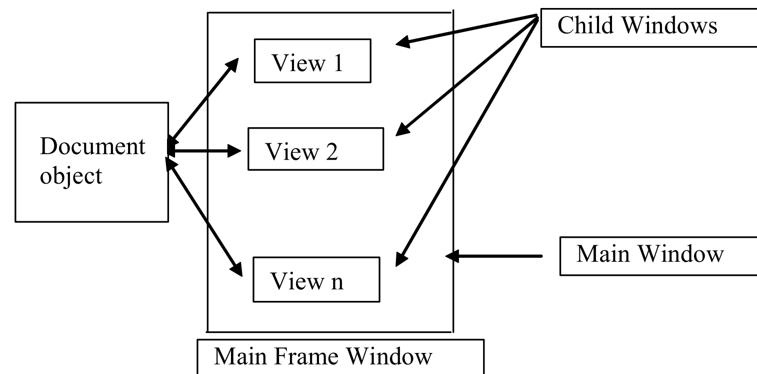


Fig. 1.17 A Multiple Document Interface Application

1.9.1 Classes for MDI Application

The classes and files that AppWizard generates for an MDI program are similar to the classes and files it generates for a Single Document Interface (SDI) program. There are however some differences in the tasks performed by these classes. Also an MDI application employs an additional class called a 'Child Frame Window' class.

The Application Class

An MDI program's application class manages the program as a whole and includes an instance member function for initializing the program.

NOTES

The Document Class

An MDI program's document class stores the document data and file I/O. An MDI program however creates a separate instance of this class for each open document rather than reusing a single instance.

The MainFrame Window Class

An MDI program's mainframe window class manages the program's mainframe window. However, rather than being derived directly from CFrameWnd MFC class, it is derived from CMDIFrameWnd. The frame window is like the main window of the application: it has a sizing border, a title bar, a window menu, a minimize button and a maximize button.

In an MDI program, the mainframe window does not create a single view window, rather it frames the general application workspace, within the application workspace in a separate 'Child Frame' window for each open document.

The class of the mainframe window is not included in the program's document template because the mainframe window is not automatically created when the first document is opened. The InitInstance function must be explicitly created and should display the cclass.

Calling the Create or LoadFrame member function of CFrameWnd can create the MDI frame window. Though the CMDIFrameWnd is derived from the CFrameWnd, a frame window class derived from the CMDIFrameWnd need not be declared with DECLARE_DYNCREATE.

The CMDIFrameWnd class inherits much of its default implementation from the CFrameWnd. For a detailed list of these features, refer to the CFrameWnd class description. The CMDIFrameWnd class has the following additional features:

- An MDI frame window manages the MDICLIENT window, repositioning it in conjunction with the control bars. The MDI client window is the direct parent of MDI the child frame windows. The WS_HSCROLL and WS_VSCROLL window styles specified on a CMDIFrameWnd, apply to the MDI client window rather than the main frame window so that the user can scroll the MDI client area (for example, as in the Windows Program Manager).
- An MDI frame window owns a default menu that is used as the menu bar when there is no active MDI child window. When there is an active MDI child window, the MDI frame window's menu bar is automatically replaced by the MDI child window menu.
- An MDI frame window works in conjunction with the current MDI child window, if there is one. For instance, the command messages are delegated to the currently active MDI child before the MDI frame window.

NOTES

- An MDI frame window has default handlers for the following standard Window menu commands:
  - ID_WINDOW_TILE_VERT
  - ID_WINDOW_TILE_HORZ
  - ID_WINDOW_CASCADE
  - ID_WINDOW_ARRANGE
- An MDI frame window also has an implementation of ID_WINDOW_NEW, which creates a new frame and view on the current document. An application can override these default command implementations to customize the MDI window handling.

```
CMainFrame * pMainFrame=  new CMainFrame;
If(pMainframe -> Loadframe/////9IDR_MAINFRAME)) returns
False
M_pMainWnd = pMainframe;

// The main window has been initialize,
// so show and update it

pMainFrame -> ShowWindow( m_nCmdShow );
pMainframe -> UpdateWindow ();
```

The first statement creates the instance of the main frame window class, “CMainFrame”. The call to the “CFrameWnd” member function ‘Loadframe” creates the main frame window itself.

The window handle is stored in the CWndApp data member m+pMainWnd, later in the InitInstance(), the call to CWnd :: Showwindowmakes the window visible and the call to CWnd :: Updatewindow causes the client area of the window to be drawn.

The CMDIMainFrame Class Members

(1) CMDIFrameWnd( );

Calls Create ( ) or LoadFrame ( ) functions to create the appliaction’s Main frame window

```
Syntax :  CMDIFrameWnd( );
```

(2) CreateClient ()

Creates a MDI client window that manages the CMDIChildWnd object.

```
Syntax : virtual BOOL CreateClient(LPCREATESTRUCT
lpCreateStruct, CMenu* pWindowMenu);
```

Where :

lpCreateStruct	:	A long pointer to a CREATESTRUCT structure.
pWindowMenu	:	A pointer to the Window pop-up menu.

(3) CreateNewChild ()

Creates a new child window

```
Syntax :CMDIChildWnd* CreateNewChild( CRuntimeClass*
pClass, UINT nResource, HMENU hMenu = NULL, ACCEL hAccel
= NULL);
```

NOTES

Where:

pClass : The run-time class of the child window to be created.

hMenu : The ID of shared resources associated with the child window.

hAccel : The child window's menu.

(4) GetWindowMenuPopup ()

Returns handle of the main pop-up menu

```
Syntax : virtual HMENU GetWindowMenuPopup(HMENU hMenuBar);
```

Where :

kMenuBar : Current menu bar.

(5) MDIActivate

Activates MDI child window.

```
Syntax :void MDIActivate( CWnd* pWndActivate );
```

Where :

pWndActivate : Pointer to the child window to be activated.

(6) MDICascade ()

Arranges all the child windows in cascade form

```
Syntax : void MDICascade( );
void MDICascade(int nType);
```

Where :

nType : Specifies a cascade flag. Only the following flag can be specified: **MDITILE_SKIPDISABLED**, which prevents disabled MDI child windows from being cascaded.

(7) MDIGetActive ()

Returns pointer to the current active child window with the flag whether that window is maximized.

```
Syntax : CMDIChildWnd* MDIGetActive(BOOL* pbMaximized =
NULL) const;
```

Where :

pbMaximized : A pointer to a BOOL return value. Set to TRUE on return if the window is maximized; otherwise FALSE.

(8) MDIIconArrange( )

Arranges all minimized child windows

```
Syntax : void MDIIconArrange( );
```

(8) MDIMaximize ( )

Maximizes the specified child window

```
Syntax : void MDIMaximize(CWnd* pWnd);
```

NOTES

Where :

pWnd : Pointer to the child window to be maximized.

(9) MDINext ( )

Activates the child window that is immediately behind the current active window and makes the current active window inactive.

```
Syntax : void MDINext( );
```

(10) MDIPrev ( )

Makes previous window active and places the current active window immediately behind it

```
Syntax : void MDIPrev( );
```

(11) MDIRestore( )

Restores the child window from maximized or minimized

```
Syntax : void MDIRestore(
   CWnd* pWnd
);
```

Where :

pWnd : Pointer to the child window to be restored.

(12) MDISetMenu ( )

Replaces the MDI mainframe window menu or pop-up menu or both.

```
Syntax : CMenu* MDISetMenu( CMenu* pFrameMenu, CMenu*
pWindowMenu );
```

Where ;

PFrameMenu : Specifies the menu of the new frame-window menu. If **NULL**, the menu is not changed.

PWindowMenu : Specifies the menu of the new Window pop-up menu. If **NULL**, the menu is not changed.

(13) MDITile ( )

Arranges all child windows in the tiled form

```
Syntax : void MDITile( );
void MDITile(
   int nType
);
```

Where :

- Type : Specifies a tiling flag. This parameter can be any one of the following flags:

NOTES

- MDITILE_HORIZONTAL: Tiles MDI child windows so that one window appears above another
- MDITILE_SKIPDISABLED: Prevents disabled MDI child windows from being tiled
- MDITILE_VERTICAL: Tiles MDI child windows so that one window appears beside another

The ChildFrame Window Class

In an MDI program, the child frame window class manages the child frame windows. Each child frame window creates a view for displaying an open document.

The child frame window class is not used in an SDI program.

Since the CChildFrame class is used for creating and managing the child frame window, the InitInastance() function includes it in the program's document template.

An MDI child window looks much like a typical frame window, except that the MDI child window appears inside an MDI frame window rather than on the desktop. An MDI child window does not have a menu bar of its own, but instead shares the menu of the MDI frame window. The framework automatically changes the MDI frame menu to represent the currently active MDI child window.

There are three different ways to create an MDI child window class.

(1) Directly constructing the MDI child window class using Create () function.
(2) Directly constructing the MDI child window class using LoadFrame.
(3) Indirectly constructing the MDI child window class using Document template.

If CMDiChildFrame contains view and document object, it must be constructed from the template instead of using a direct method.

The CDocTemplate object orchestrates the creation of the frame, the creation of the containing views and the connection of the views to the appropriate document. The parameters of the CDocTemplate constructor specify the CRuntimeClass of the three classes involved (document, frame and view). A CRuntimeClass object is used by the framework to dynamically create new frames when specified by the user (for example, by using the File New command or the MDI Window New command).

A frame-window class derived from CMDIChildWnd must be declared with DECLARE_DYNCREATE in order for the above RUNTIME_CLASS mechanism to work correctly.

The CMDIChildWnd class inherits much of its default implementation from the CFrameWnd. For a detailed list of these features, please refer to the CFrameWnd class description. The CMDIChildWnd class has the following additional features:

- In conjunction with the CMultiDocTemplate class, multiple CMDIChildWnd objects from the same document template share the same menu, saving Windows' system resources.

NOTES

- The currently active MDI child window menu entirely replaces the MDI frame window's menu, and the caption of the currently active MDI child window is added to the MDI frame window's caption. For further examples of the MDI child window functions that are implemented in conjunction with an MDI frame window, see the CMDIFrameWnd class description.

The CMDIChildframe Members

(1) CMDIChildFrame () Constructor

Creates CMDIChildFrame object

```
Syntax :CMDIChildWnd( );
```

(2) Create () Function

Creates a CMDIChildFrame object

```
Syntax : virtual BOOL Create( LPCTSTR lpszClassName,
LPCTSTR lpszWindowName, DWORD dwStyle = WS_CHILD |
WS_VISIBLE | S_OVERLAPPEDWINDOW, const RECT& rect =
rectDefault, CMDIFrameWnd* pParentWnd = NULL,
CCreateContext* pContext = NULL);
```

Where :

LpszClassName	:	Any class neame registered using AfxregisterWndClas, NULL in case of standard CMDICHildWnd.
LpszWindowName	:	Null terminated string represents window name to be displayed in the title bar.
DwStyle	:	Shows window style, WS_CHILD in case CMDIChildFrame.
Rect	:	Size of the Child window.
PParentWnd	:	Pointer to the parent window , if Null main application window is treated as parent window.
PContext	:	This parameter is NULL.

(3) GetMDIFrame ( )

Returns pointer to the MDI parent frame

```
Syntax :  CMDIFrameWnd *  (3( GetMDIFrame ( );
```

(4) MDIActivate( )

Activates MDI child window irrespective to the main window

```
Syntax : void MDIActivate( );
```

(5) MDIDestroy( );

Destroys the child window

```
Syntax : void MDIDestroy( );
```

(6) MDIMaximize( );

Call this member function to maximize an MDI child window

```
void MDIMaximize( );
```

(7) MDIRestore( );

Restores the MDI child window from maximized and minimized

```
Syntax : void MDIRestore( );
```

1.9.2 The View Class

The view class in an MDI program is used for creating and managing the view window that displays each open document.

The view window for each document occupies the client area of the document's child frame window.

Check Your Progress

23. What do you understand by the messages?
24. Define a Multiple Document Interface (MDI).
25. Where is the child view object in the MDI application created?
26. What is an application class?
27. What is the function of an MDI program's mainframe window?
28. What is the function of a child frame window?
29. What are the various ways for creating an MDI child window class?

1.10 ANSWER TO 'CHECK YOUR PROGRESS'

1. Microsoft Developer Studio is the centre of the Visual C++ development environment. It is used to integrate the development tools and the Visual C++ compiler.
2. With the help of Visual C++ and Developer Studio, it is very simple to create Windows applications by using tools and Wizards provided by the Development Studio with MFC class library.
3. The Developer Studio tools are as follows:
   - Developer Studio Code Editor
   - Resource Editor
   - Integrated Debugger
4. AppWizard supports the following three types of applications:
   (i) Single document applications based on the document VIeww architecture.
   (ii) Multiple document applications based on the document VIeww architecture.

NOTES

(iii) Dialog box-based applications in which dialog box serves as the application's main window.

5. Microsoft Foundation Class enables us to write Windows application using C++.
6. InfoViewer has several advantages over hard copy of documentation.
   - It is fully searchable.
   - Annotations and bookmarks can be added in the documentation.
   - Context sensitive help can be used pressing F1 key.
   - It is completely integrated into Developer Studio.
   - InfoViwer documentation can be printed where hard copy is required.
7. OpenWorkspace displays the Open File Dialog Box, which allows to select a workspace file (.MDP) to be opened. This closes any project workspace currently open and opens the selected project, which becomes the current project in the Developer Studio.

   Project Workspace performs the following functions:
   - It stores the types of applications that are created.
   - It allows the Developer Studio to keep track of all elements that go into the application.
   - It also stores all compiler and linker settings, so it is not required to reset them every time the project is reloaded.
8. Resource View: This view allows you to edit as well as to find the various resources in your application. It also consists of various icons, menus, dialog window designs, etc.

   File View: This view allows you to display the file views which are associated with your application. It also navigate all the files that make up your application.

   Class View: The class view displays the class level view of your source code. It allows you to manipulate and navigate the source code also.
9. The Wizard Bar toolbar allows to execute a number of Class Wizard actions without opening the Class Wizard.
10. If we use Windows Explorer, then it will display three files:
    - A file having an extension .ncb that records data about Intellisense. Here the Intellisense is the ability that offers prompting and auto-completion for code in the Editor window as you enter it.
    - A file having extension .suo that records the user options that apply to the solution.
    - A file having extension .sln that records information about the projects in the solution.
11. There are two types of dialog boxes—modal dialog box and modeless dialog box.

NOTES

12. The WM_INITDIALOG message activates the OninitDialog() handler. In this function, the necessary initialization is done to prepare the controls for action.
13. Controls are basically user interface objects, such as push buttons, list boxes, scroll bars, etc. They are used to supply inputs.
14. Button controls are of four types: Push buttons, Check boxes, Radio buttons and Group boxes.
15. A standard list box displays strings in a vertical column and allows only one item to be selected at a time.
16. Static controls contain the data which does not change during the execution of the program.
17. Combo boxes are of three types: (i) A simple combo box, (ii) A drop-down combo box and (iii) A drop-down list box.
18. The Edit control has built-in support for operations like cut, copy, paste, undo, etc.
19. A scroll bar control can be placed anywhere in the windows and can be of any reasonable height or width.
20. The first parameter dwStyle is the style of the windows control. The second parameter specifies the size of the control. The third parameter pParentWnd is a pointer to the owner of the control. The last parameter nID is the control ID by the parent to communicate with the control.
21. The LBS_SORT function forces the listbox to sort all the incoming strings alphabetically as it adds them to the list. This means that the strings are arranged in a different order than the one in which they were sent.
22. The list box sends the notification messages to its owner window.
23. Events in a Windows application are signal or action received by the program such as the user press the key or click the mouse or timer reaching zero. The Windows operating system records every event in a message and places the message in the program for which the message is proposed. So, a Windows message is record of the data which is related to the event. The message queue for an application is a sequence of messages waiting to be managed by the application.
24. A Multiple Document Interface (MDI) is a standard way to write an application in which one master window holds a number of child windows.
25. The child view object in the MDI application is created inside the Childframe object, which itself is contained in the application's Mainframe object.
26. An MDI program's application class manages the program as a whole and includes an instance member function for initializing the program.
27. An MDI program's mainframe window manages the program's main frame window. It has a sizing border, a title bar, a window menu, a minimize button and a maximize button.
28. A child frame window creates a view for displaying an open document.

NOTES

29. There are three different ways of creating a MDI child window class. They are as follows:
    - By directly constructing the child window class using Create ( ) function.
    - By directly constructing the child window class using Loadframe.
    - By indirectly constructing the child window class using Document template.

1.11 SUMMARY

- Microsoft Visual C++ is a tool for building and debugging Windows based applications and libraries in an integrated windows environment. Visual C++ makes it much easier to handle complex job of developing applications for Windows by incorporating the integrated Windows-based environment with high level C++ application classes.
- Development Studio is used to integrate the development tools and the Visual C++ compiler. A Windows program can be created and scanned through an impressive amount of online help and debugged without having developer studio.
- Developer Studio tools are used for complete project management. It is used to link variety of code modules into one project, which is then used as unit for building applications. Developer Studio tools are:
    - o Developer Studio Code Editor
    - o Resource Editor
    - o Integrated Debugger
- Developer studio includes two wizards to develop Windows applications: AppWizard and ClassWizard.
- The Microsoft foundation class enables us to write Windows application using C++. The class library consists of C++ classes that represent an ‘application frame work’. The classes are designed to be used together to create working skeleton application that provides much of the user interface and functionality that users expect in a Windows application.
- InfoViewer is the outline help system integrated into developer Studio. It is the only documentation included with the product because Visual C++ is not sold with a documentation.
- The Project Workspace controls files are included in the application and governs how they are combined to build the finished application. Its other functions are as follows:
    - o It stores the types of applications that are created.
    - o It allows the Developer Studio to keep track of all elements that go into the application.
    - o It allows the facility to compile and link only those modules that have changed since the project was built last time.

NOTES

- o It also stores all compiler and linker settings, so it is not required to reset them every time the project is reloaded.

- In Windows environment, everything is shared—the screen, the keyboard, the mouse, even the user. The program written for Windows must cooperate with Windows and with other programs that may be running at the same time.
- Resource view allows you to edit as well as to find the various resources in your application. It also consists of various icons, menus, dialog window designs, etc.
- The Wizard Bar toolbar which offers to execute a number of Class Wizard actions without opening the Class Wizard.
- Dialog boxes are popup windows, which simultaneously combine several child window controls on their surface. The creation and use of controls is much simpler on the dialog box than on the window.
- Dialog boxes are used to enter the user's inputs to the application through controls like text box, button control, edit box, selecting options, checking values etc.
- Controls are basically user interface objects. They could be a push button, list boxes, scroll bars, etc. They are used to supply inputs. They are often known as the child windows of the mainframe window.
- There are four types of button controls: Push buttons, Check boxes, Radio buttons and Group boxes. A button can be clicked with the left mouse button or the spacebar.
- A list box is used to display a list of strings called items. It is of different types like a string, a file name, an address, a roll number, etc.A list box could have multiselected items and multiple columns.
- A combo box is a combination of a single line edit control and a list box. It is of three types: simple combo box, drop-down combo box and drop-down list box.
- The most common use of a list box is in the dialog boxes. A list box typically allows the user to select a filename, directory, and so on.
- The MFC wraps the standard Window list box message into a ClistBox class method. It usually needs to handle only the notification messages. The list box then sends the notification messages to its owner window.
- A Multiple Document Interface (MDI) is a standard way to write an application in which one master window holds a number of child windows.
- An MDI program's application class manages the program as a whole and includes an instance member function for initializing the program.

1.12 KEY TERMS

- **Bitmap:** It is a kind of memory organization or image file format which is used to store digital images.

- **Developer Studio Editor:** It is a tool which not only provides text formatting but also provides features that help to write source code easily.
- **Appwizard:** It is an application development tool is used to create the basic outline of the windows application.
- **Classwizard:** It is an application development tool is used to define the class in a program created with appwizard.
- **Infoviewer:** Infoviewer is the outline help system, integrated into developer studio.
- **Dialog Box:** They are popup windows which simultaneously combine several child window controls on their surface.
- **Controls:** They are basically user interface objects, such as push buttons, list boxes, scroll bars, etc.
- **List Box Control:** It is a control used to display a list of text strings called items.
- **Static Controls:** They refer to the controls that contain data which does not change during execution of the program.
- **Combo Box:** It is a combination of a single line edit control and a list box.
- **Multiple Document Interface (MDI)**: It is a standard way to write an application in which one master window holds a number of child windows.

1.13 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Which function is used to control the size of Window during creation?
2. How will you write the VC++ (Win32) application?
3. Define a dialog box.
4. What are the types of dialog box?
5. In how many ways does an application take input?
6. What is the difference between placing control on the window screen and on the dialog box?
7. What is the difference between a list box control and a combo box control?
8. What is the difference between check box and radio button?
9. What is the difference between SDI application and MDI application?

Long-Answer Questions

1. What parameters are passed to WinMain() and what does they signify?
2. Discuss about the structure of a VC++ application with the help of examples.
3. Explain about the sample VC++ (Win32) application. Give appropriate examples.

NOTES

4. Describe the process involved in the initialization of controls within the dialog box(WM_INITDIALOG).
5. What are controls? Describe the types of Button controls.
6. Explain the process of creation of the windows controls.
7. Enumerate the ListBox styles and the meaning of each style.
8. What are the different classes an MDI application? Explain.

1.14 FURTHER READING

Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.

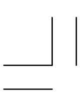
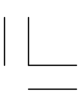
Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.

Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.

Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6*, 1st Edition. New Delhi: BPB Publications.

Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.

Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 2 DRAWING ON THE SCREEN, PRINTING AND FILE HANDLING

NOTES

Structure

2.0 Introduction
2.1 Objectives
2.2 Drawing on the Screen
  2.2.1 Device Contexts
  2.2.2 Device Objects
  2.2.3 Wizard Support for Device Context
  2.2.4 Stock Objects
  2.2.5 A DC Example
  2.2.6 Using Color in Windows Applications
2.3 Printing and Print Preview
  2.3.1 MFC Printing Apple Cation
  2.3.2 Adding Functionalities to MFC Print
2.4 Persistence and File I/O
  2.4.1 Basic File Operations
  2.4.2 Files and Windows Applications
  2.4.3 Serialization
2.5 Answer to 'Check Your Progress'
2.6 Summary
2.7 Key Terms
2.8 Self-Assessment Questions and Exercises
2.9 Further Reading

2.0 INTRODUCTION

The Windows OS provides several levels of abstraction. These levels are used for creating and drawing using graphics in the applications. In DOS programming, there is requirement to exercise a better control over the graphics hardware, which is used to draw any type of image in an application. All this process requires widespread knowledge as well as understanding of the several types of graphics cards which might be in their computers. There were graphics libraries also which can be used for the applications. It was fairly vigorous programming to add this ability to the applications.

Printing is one of the hardest things for accurately implementing in your Win32 program. MFC supports printing and print preview of your program documents using C View class. You get print preview, standardized dialogs like page setup, print setup, print job interruption dialog. The MFC source files are the better resource for help regarding the printing and print preview. But sometimes the MFC hides most of the features in members of several classes. Files are very important for storing information permanently. Information is stored in files for data processing by programs. File I/O is reading from and writing to files.

In this unit, you will learn about the drawing on the screen, printing and print preview, persistence and file I/O, basic file operations, files and windows applications and serialization.

NOTES

2.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss about the drawing on the screen
- Explain about the printing and print preview of program documents
- Elaborate on the persistence and file I/O
- Discuss the various file operations
- Understand the files and windows application
- Discuss how to scrialize the document

2.2 DRAWING ON THE SCREEN

The Windows OS provides several levels of abstraction. These levels are used for creating and drawing using graphics in the applications. Earlier, while using DOS programming, there is requirement to exercise a better control over the graphics hardware, which is used to draw any type of image in an application. All this process requires widespread knowledge as well as understanding of the several types of graphics cards may be available in their computers. There were graphics libraries also which could be bought for the applications. It was fairly vigorous programming to add this ability to the applications. Microsoft Windows has made the task very easy. A virtual graphics device for all the Windows application is provided by Microsoft. And this virtual graphics device doesn't change with the hardware, instead remains the same for all graphics hardware. This stability provides you the ability to create any type of graphics that you require in your applications.

2.2.1 Device Contexts

The device context consists of the information about the application, the system and the window in which you are drawing any type of graphics. In this context a graphic is being drawn, where it is currently located on the screen, to know about how much the area is visible. While drawing the graphics, you must take care that it always draw them in the context of an application window.

The window may be minimized, full view, completely hidden or partly hidden. You can draw the graphics on the window using its device context. Windows helps to keep track of each device context. Using windows, it determines about what part and how much of the graphics you draw to really display for the user. The device context which you are using to display the graphics is the visual context of the window where you are drawing it.

Mainly two resources are used by the device context to accomplish most of its drawing and graphics functions. The names of these two resources are pens and brushes. These pens and brushes are used to complete similar however different tasks. The device context uses pens to draw shapes and lines and the brushes paint areas of the screen. It is similar as working on paper while you are using a pen to draw an outline of an image and with the help of paintbrush, you are filling the color in between the lines.

NOTES

The Device Context Class

In VC++, the MFC device context class (CDC) provides various functions for drawing squares, lines, circles, curves etc. All the functions use the device context information to draw on the application windows, so these functions are part of the device context class. With the help of a pointer, you can create a device context class instance to the window class, which you want to associate with the device context. Due to this, the device context class place all of the code related with freeing and allocating a device context in the class constructor and destructors.

The Pen Class

In the pen class, CPen, used to identify the width and color width for drawing lines on the screen. It is the primary resource tool for drawing any type of line on the screen. When an instance of the CPen class is created, you have to specify the line color, type and thickness. After creating a pen, you can select and use it as the current drawing tool for the device context. It is used for all of the drawing commands to the device context. To create a new pen, select it as the current drawing pen. The code given below is for this pen class:

```
// Create the device context
CDC dc(this);
// Create the pen
CPen lPen(PS_SOLID, 1, RGB(0, 0, 0));
// Select the pen as the current drawing pen
dc.SelectObject(&lPen);
```

There are various types of pen styles. These pen styles are used to draw different patterns while drawing lines. Figure 2.1 shows different types of pen styles that can be used in your applications with any color.

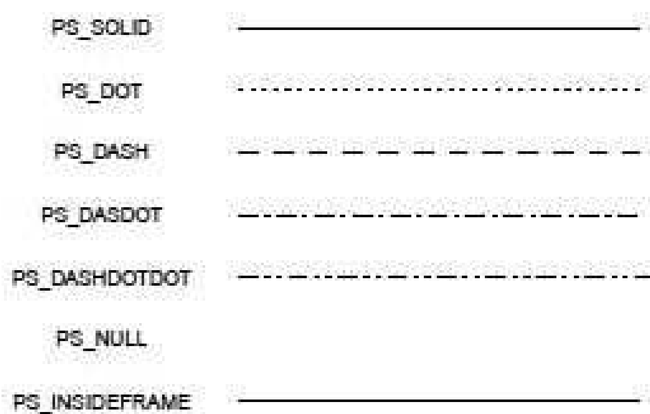


Fig. 2.1 *Windows Pen Styles*

With these line styles, you have to also identify the pen's color and width. The combination of these three variables specifies the presence of the resulting lines. The range of the line width can be from 1 on up, instead if you measure a width of 32, it is hard to exercise any level of accuracy in the drawing efforts.

You can specify the color as a RGB value. This RGB has three separate values for the brightness of the red, green, and blue color components of the

NOTES

pixels which are visible on the computer screen. The range of these separate values can be from 0 to 255. The RGB function also combines them into a single value in the format required by Windows. The most common colors are shown in Figure 2.2.

Color	Red	Green	Blue
Black	0	0	0
Blue	0	0	255
Dark blue	0	0	128
Green	0	255	0
Dark green	0	128	0
Cyan	0	255	255
Dark cyan	0	128	128
Red	255	0	0
Dark red	128	0	0
Magenta	255	0	255
Dark magenta	128	0	128
Yellow	255	255	0
Dark yellow	128	128	0
Dark gray	128	128	128
Light gray	192	192	192
White	255	255	255

Fig. 2.2 *Most Common Window Colors*

The Brush Class

The brush class, CBrush, is used to create brushes which are used to define how areas will be filled in. While drawing the shapes that are enclosed in the area and fill in, where the outline is drawn with the help of current pen. The inside area is filled with the help of current brush. Brushes can be a pattern of lines, solid colors or a repeated pattern created from a small bitmap.

To create a solid-color brush, you require to specify the color to use as given below:

```
CBrush lSolidBrush(RGB(255, 0, 0));
```

To create a pattern brush, you require to specify not only the color, but also the pattern to use:

```
CBrush lPatternBrush(HS_BDIAGONAL, RGB(0, 0, 255));
```

After creating a brush, you can select it with the help of device context object, same as with the pens. After selecting the brush, it is used as the current brush when you draw something that uses a brush. With the help of pens, you can select a number of standard patterns, as shown in Figure 2.3.

An additional style of brush, HS_BITMAP uses a bitmap as the pattern for filling the stated area. The size of this bitmap is limited in size to 8 pixels by 8 pixels (8 x 8). This is a smaller bitmap which is normally used for small images and toolbars. If you are using a larger bitmap, then it takes only the upper-left corner that is limiting it to an 8-by-8 square.

NOTES

By creating a bitmap resource, you can create a bitmap brush for your application and assigning it an object ID. After doing this, you can create a brush with the help of code given below:

```
CBitmap m_bmpBitmap;
// Load the image
m_bmpBitmap.LoadBitmap(IDB_MYBITMAP);
// Create the brush
CBrush lBitmapBrush(&m_bmpBitmap);
```

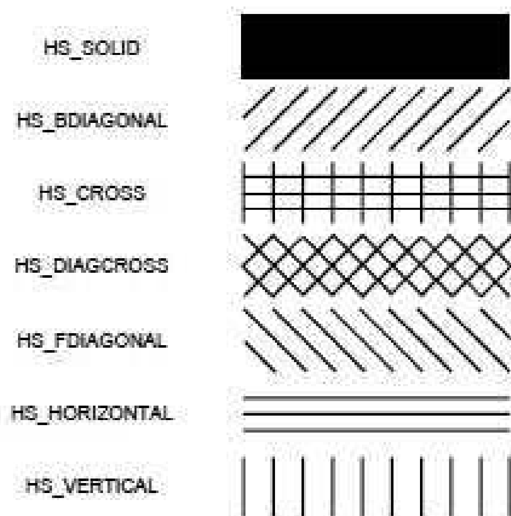


Fig. 2.3 *Standard Brush Patterns*

The Bitmap Class

To display images in your applications, it is require that you have a couple of options. You can add fixed bitmaps to the application. You add them as resources with object IDs assigned to them and use as static picture controls or an ActiveX control that displays images. Another option of this bitmap class is CBitmap, which is used to exercise complete control over the image display.

If you use the bitmap class, you can load bitmap images from files on the system disk. You can resize the images as required to make them fit in the space. To add the bitmap as a resource, create an instance of the CBitmap class by using the resource ID of the bitmap as the image to be loaded.

To load a bitmap from a file, use the LoadImage API call. After loading the bitmap, you can use the handle for the image to attach the image to the CBitmap class with the code given below:

```
// Load the bitmap file
HBITMAP hBitmap =
(HBITMAP)::LoadImage(AfxGetInstanceHandle(),
m_sFileName, IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE |
LR_CREATEDIBSECTION);
// Attach the loaded image to the CBitmap object.
m_bmpBitmap.Attach(hBitmap);
```

You can create a second device context and select the bitmap after loading the bitmap into the CBitmap object. After creating the second device context, it is required to make it compatible with the primary device context. This is done before the bitmap is selected. As we know that, the device contexts are created by the OS for a particular output device. It must be ensured that the second device context is also attached to the same output device as the first. The code given below is for creating a device context:

```
// Create a device context
CDC dcMem;
// Make the new device context compatible with the real
DC
dcMem.CreateCompatibleDC(dc);
// Select the bitmap into the new DC
dcMem.SelectObject(&m_bmpBitmap);
```

This can copy the bitmap into the regular display device context when the bitmap is selected into a compatible device context. This is done by using the BitBlt function. The code to copy the bitmap to the display DC is given below:

```
// Copy the bitmap to the display DC
dc->BitBlt(10, 10, bm.bmWidth,
bm.bmHeight, &dcMem, 0, 0, SRCCOPY);
// Resize the bitmap while copying it to the display DC
dc->StretchBlt(10, 10, (lRect.Width() - 20),
(lRect.Height() - 20), &dcMem, 0, 0,
bm.bmWidth, bm.bmHeight, SRCCOPY);
```

With the help of StretchBlt function, you can resize the bitmap. This is required to fit in any area on the screen.

2.2.2 Device Objects

There are two types of device objects that are: CClientDC and CWindowDC. As we know that, a window's client area eliminates the menu bar, the border and the caption bar. If a *CClientDC* object is created, then there is a device context which is mapped only to this client area. This can't be drawn outside it. The point (0, 0) generally states the upper-left corner of the client area. An MFC *CView* object relates to the child window which is enclosed into a separate frame window, often along with scroll bars, a toolbar and a status bar. The client area of the view doesn't consists these other windows.

If the window consists of a docked toolbar along the top. Consider an example, (0, 0) indicates to the point nearly under the left edge of the toolbar. If you create an object of class *CWindowDC*, then the point (0, 0) is at the upper-left corner of the non-client area of the window. With this whole-Window device context, you can draw in the window's border, in the caption area.

Constructing and Destroying CDC Objects

After constructing a *CDC* object, destroying is also important when you're done with it. Microsoft Windows limits the number of available device contexts, and if you fail to release a Windows device context object, then a small amount of memory

is vanished until the program exits. You will create a device context object inside a message handler function like *OnLButtonDown*. The easiest method is that the device context object is destroyed to construct the object on the stack. The code is given below:

```
void CMyView::OnLButtonDown(UINT nFlags, CPoint point)
{
CRect rect;    CClientDC dc(this);
// constructs dc on the stack    dc.GetClipBox(rect);
// retrieves the clipping rectangle
}
// dc automatically released
```

The *CClientDC* constructor has a window pointer as a parameter. Whereas the destructor for the *CClientDC* object is called when the function returns. Another method to get a device context pointer is using the *CWnd::GetDC* member function, as shown in the code given below:

```
void CMyView::OnLButtonDown(UINT nFlags, CPoint point)
{
CRect rect;    CDC* pDC = GetDC();
// a pointer to an internal dc
pDC->GetClipBox(rect);
// retrieves the clipping rectangle
ReleaseDC(pDC);
// Don't forget this
}
```

State of the Device Context

A device context is necessary for drawing. When you use a *CDC* object to draw an ellipse, what you see on the screen depends on the current "state" of the device context. This state of the device context consists of the following:

- o Several details like polygon filling mode and text alignment parameters
- o Attached GDI drawing objects like fonts, brushes and pens
- o The mapping mode that regulates the scale of items when they are drawn

Consider an example selecting a gray brush prior to drawing an ellipse that results in the ellipse which is having a gray interior. After creating a device context object, it has definite default characteristics, like a black pen which is used for shape boundaries. All other state characteristics are allocated with the help of CDC class member functions. In the device context, GDI objects are selected by means of the overloaded *SelectObject* functions. For example, a device context can have one brush, one pen or one font selected at a given time.

The CPaintDC Class

If you override your view's *OnPaint* function, then you will require the *CPaintDC* class only. The default *OnPaint* calls *OnDraw* with a properly set up device context. Sometimes, it is required to display-specific drawing code. The code given below is for *OnPaint* function that creates a *CPaintDC* object:

NOTES

```
void CMyView::OnPaint()
{
CPaintDC dc(this);
OnPrepareDC(&dc);
// explained later
 dc.TextOut(0, 0, "for the display, not the printer");
OnDraw(&dc);
// stuff that's common to display and printer
}
```

2.2.3 Wizard Support for Device Context

There are two types of context-sensitive help implemented in Windows applications. The first one is using "F1 Help" that consists of launching WinHelp with the suitable context based on the currently active object. The second one is "Shift+ F1" mode. In this mode, the user proceeds to click on an object and the mouse cursor changes to the help cursor. WinHelp is launched to provide help for the object that the user clicked on.

The Microsoft Foundation Classes implement both of these forms of help. The framework supports two simple help commands: Using Help and Help Index. The Microsoft Foundation classes accept a single Help file. This Help file must have the same path and name as the application. For example, if the executable file is C:\MyApplication\MyHelp.exe then help file must be C:\MyApplication \MyHelp.hlp. There are two simple help commands which are implemented by the Microsoft Foundation Classes:

- ID_HELP_USING which is implemented by CWinApp::OnHelpUsing
- ID_HELP_INDEX which is implemented by CWinApp::OnHelpIndex

The first command shows the user help on using the WinHelp program. The second one shows the Help index for the application. The F1 key is mostly translated to a command with the help of an ID of ID_HELP by an accelerator placed into the main window's accelerator table. Instead of how the ID_HELP command is produced, it is transmitted as a normal command until it reaches a command handler. CWinApp::OnHelp efforts to launch WinHelp in the following order:

1. First check for an active AfxMessageBox call with a Help ID. Check if message box is currently active, then WinHelp is launched with the context that is suitable to that message box.
2. A WM_COMMANDHELP message sends to the active window and if that window doesn't not respond by launching WinHelp, then the same message is sent to the ancestors of that window.
3. Sends a ID_DEFAULT_HELP command to the main window, which raises the default Help. This command is mapped to CWinApp::OnHelpIndex.

The help mode (Shift +F1) is another mode of context-sensitive Help. This mode is entered with the help by pressing SHIFT+F1. It is implemented as a command (ID_CONTEXT_HELP). While entering this mode, the help mouse cursor is exhibited over all areas of the application. Instead, if the application would generally

display its own cursor for that area. The user is able to use the keyboard or mouse to select a command. Instead of executing the command, Help on that command is shown.

The user can click a visible object on the screen, for example like a button on the toolbar or Help will be shown for that object. CWinApp::OnContextHelp provide this mode of Help. All keyboard input is inactive during the execution of this loop, except for keys that access the menu. The command translation is still achieved with the help of PreTranslateMessage to let the user to receive help on that command or to press an accelerator key.

If there are specific actions or translations taking place in the PreTranslateMessage function which shouldn't take place during SHIFT+F1 Help mode, then you should check the *m_bHelpMode* member of CWinApp before executing those operations. The CDialog implementation of PreTranslateMessage always checks this before calling IsDialogMessage. This inactivates the dialog navigation keys on modeless dialogs during SHIFT+F1 mode. During this loop, the CWinApp::OnIdle is called. If the user selects a command from the menu, which is handled as help on that command. If the user clicks a visible area of the applications window, a determination is made as to whether it is a client click or a non-client click.

OnContextHelp manages the mapping of non-client clicks to client clicks automatically and if it is a client click, then sends a WM_HELPHITTEST to the window which was clicked. If a nonzero value returns by the window, then that value is used as the context for help. If window returns zero, then OnContextHelp uses the parent window. If a Help context cannot be found, then the default is to send a ID_DEFAULT_HELP command to the main window and that was usually mapped to CWinApp::OnHelpIndex.

2.2.4 Stock Objects

A Windows GDI object type is denoted by an MFC library class. For the GDI object classes, the *CGdiObject* is the abstract base class. A Windows GDI object is signified by a C++ object of a class derived from *CGdiObject*. Following are the GDI derived classes:

- ***CFont***: It contains the complete collection of characters of specific typeface and specific size. Fonts are mostly stored on disk like device-specific or as resources.
- ***CPen***: CPen is a tool for drawing shapes and lines borders. The pen's color can be specified easily, as well as its thickness, whether it draws dotted, solid or dashed lines.
- ***CBitmap***: CBitmap is a bitmap of an array of bits, where one or more bits related to every display pixel. The bitmaps can be used to signify images or it can be used them to create brushes.
- ***CPalette:*** CPalette is a color mapping interface which allows an application to have the full benefit of the color capability of an output device. This is done without interfering with other applications.

- ***CRgn***: CRgn is a region or an area which consists of the shape like an ellipse or a polygon or combination of polygons and ellipses. These regions can be used for clipping, filling and mouse hit-testing.
- ***CBrush:*** Cbrush is used to describe a bitmapped pattern of pixels, which is used to fill the areas with the help of colors.

2.2.5 A DC Example

Device independent drawing in Windows is allowed by the Device contexts. The Device contexts can be used to draw to the metafile, to the screen or to the printer. The CDC is the most important class to draw in MFC. The member functions are provided by CDC object which is used to complete the basic drawing steps, as well as members for working with a display context related with the client area of a window.

Lines

Step 1: Consider an example of creating a new MFC based single document project with **MFCGDIDemo** name.

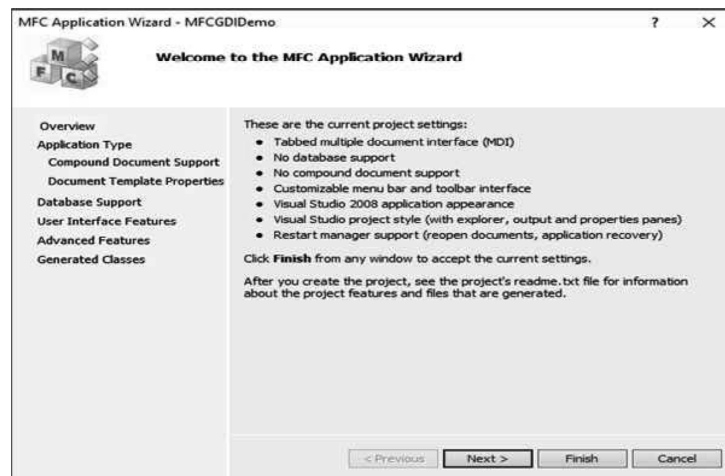


Fig. 2.4 *MFC Application Wizard Welcome Screen*

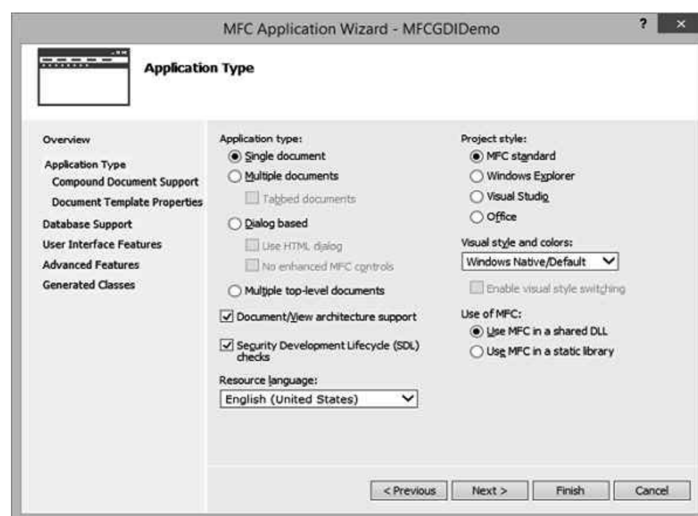


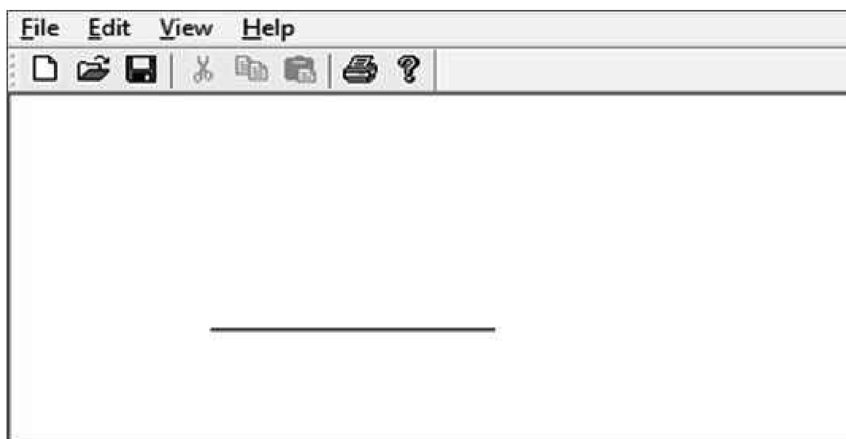
Fig. 2.5 *MFC Application Wizard*

Step 2: After creating the project, go the Solution Explorer and double click on the **MFCGDIDemoView.cpp** file under the Source Files folder.

Step3: Using the code given below, draw the line in **CMFCGDIDemo View::OnDraw()** method.

```
Void CMFCGDIDemo View:OnDraw(CDC* pDC)
{
pDC->MoveTo(94, 124),
pDC->LineTo(232, 124);
CMFCGDIDemoDoc* pDoc= GetDocument();
ASSERT_VALID(pDoc);
If(!pDoc)
Return;
//TODO: add draw code for native data here
```

Step 4: After running this application, the output will be as shown below.



Step 5: To set the starting position of a line, CDC::MoveTo() method is used. After using LineTo(), the program starts from the MoveTo() point to the LineTo() end.

Polylines

A Polyline is a series of connected lines. The lines are stored in an array of CPoint or POINTvalues. You can use the CDC::Polyline() method to draw a polyline. At least two points are required to draw a polyline. If you describe more than two points, then each line after the first would be drawn from the previous point to the next point until all points have been encountered.

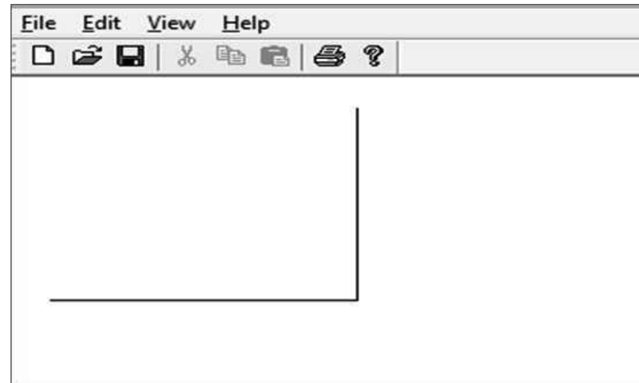
Step 1: Use the following code as an example to draw the polylines:

```
Void CMFCGDIDemo View:OnDraw(CDC* pDC)
{
CPoint Pt[6];
Pt[0]=CPoint(18, 148);
Pt[1]=CPoint(178, 148);
Pt[2]=CPoint(178, 18);
pDC=Polyline(Pt, 2);
CMFCGDIDemoDoc" pDoc = GetDocument();
```

NOTES

```
ASSERT_VALID(pDoc);
If(!pDoc)
Return;
//TODO: add draw code for native date here
}
```

Step 2: After running the application, the output will be as shown below.



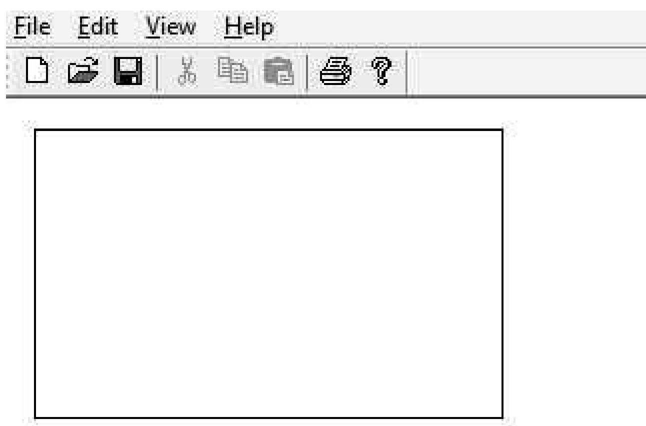
Rectangles

A Rectangle is a geometric figure which is made up of four sides. It consists of four right angles. To draw a rectangle, like the line it must be defined from where it starts and where it ends. Use the CDC::Rectangle() method to draw a rectangle.

Step 1: Use the following code to draw a rectangle.

```
void CMFCGDIDemoView::OnDraw(CDC* pDC)
{
  pDC->Rectangle(15, 15, 250, 160);
    CMFCGDIDemoDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if (!pDoc)
     return;
  // TODO: add draw code for native data here
}
```

Step 2 You will see the following output after running the application.



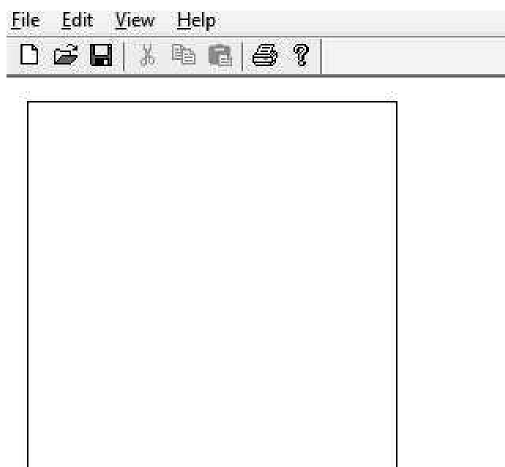
NOTES

Squares

A square is a geometric enclosed figure which is made of four equal sides and every angle is of 90^0. Use the following code as an example to draw a square.

```
void CMFCGDIDemoView::OnDraw(CDC* pDC)
{
  pDC->Rectangle(15, 15, 250, 250);
     CMFCGDIDemoDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
   if (!pDoc)
      return;
   // TODO: add draw code for native data here
}
```

When you run this application, you will see the following output:



Pies

A pie is a fraction of an ellipse. It surrounded by two lines that extent from the center of the ellipse to one side each. You can use the CDC::Pie() method to draw a pie. The syntax is given below:

```
BOOL Pie(int x1, int y1, int x2, int y2, int x3, int y3,
int x4, int y4);
```

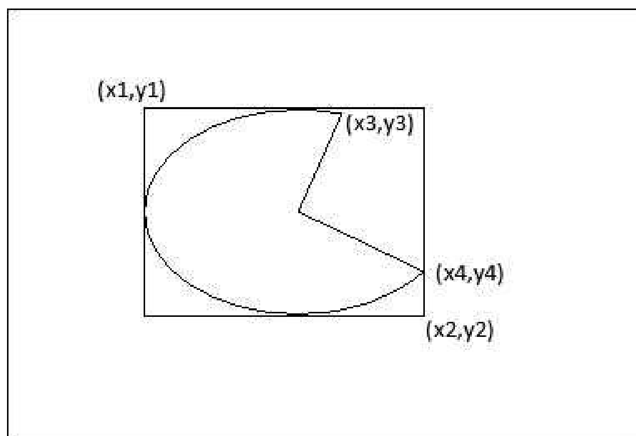


Fig. 2.6 *Pie with Upper-Left Corner and Bottom-Right Corner*

NOTES

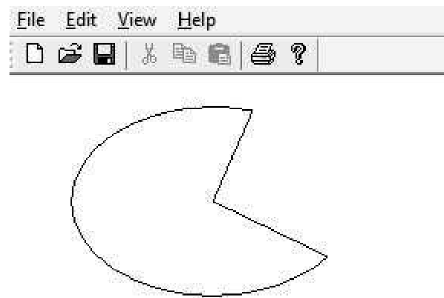
Following are some of the points related to pie.

1. The (x1, y1) is the point on the upper-left corner of the rectangle in which the ellipse that represents the pie fits. The bottom-right corner of the rectangle is (x2, y2).
2. The (x3, y3) point identifies the starting corner of the pie in a default counter clockwise direction.
3. The co-ordinate of the end point of the pie is (x4, y4).

Consider an example using the code given below to draw a pie:

```
void CMFCGDIDemoView::OnDraw(CDC* pDC)
{
    pDC->Pie(40, 20, 226, 144, 155, 32, 202, 115);
        CMFCGDIDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: add draw code for native data here
}
```

When you run the application, you will see the following output:



Arcs

An arc is a smooth curve joining two endpoints. In general, an arc is a segment or portion of an ellipse. You can use the CDC::Arc() method to draw an arc. The syntax is given below:

```
BOOL Arc(int x1, int y1, int x2, int y2, int x3, int y3,
int x4, int y4);
```

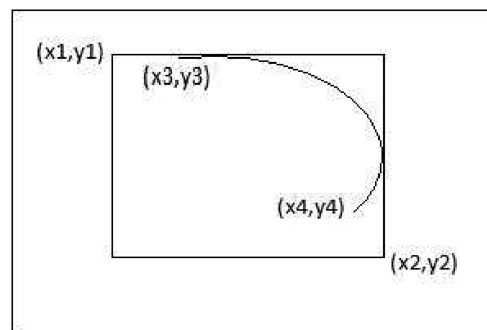


Fig. 2.7 Arc with Starting and Ending Point

The CDC class is equipped with the SetArcDirection() method. The syntax for the CDC class having the SetArcDirection() method is given below:

```
int SetArcDirection(int nArcDirection)
```

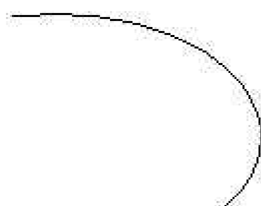

If the value and orientation is AD_CLOCKWISE then the figure drawn will be clockwise. And if the value and orientation is AD_COUNTERCLOCKWISE then the figure drawn will be counter clockwise.

I

Consider an example using the code given below to draw an arc.

```
void CMFCGDIDemoView::OnDraw(CDC* pDC)
{
  pDC->SetArcDirection(AD_COUNTERCLOCKWISE);
  pDC->Arc(20, 20, 226, 144, 202, 115, 105, 32);
    CMFCGDIDemoDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if (!pDoc)
     return;
  // TODO: add draw code for native data here
}
```

When you run this application, you will see the following output.



Chords

A chord is an arc which is having two ends and these are connected by a straight line. You can use the CDC::Chord() method to draw a chord. The syntax of Chord() method is given below.

```
BOOL Chord(int x1, int y1, int x2, int y2, int x3, int y3,
int x4, int y4);
```

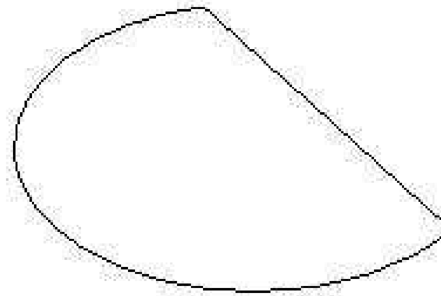
Consider an example using code given below to draw a chord.

```
Void CMFCGDIDemoView:OnDraw(CDC*pDC)
{
pDC->SetArcDirection(AD_CLOCKWISE);
pDC->Chord(20, 20, 226, 144, 202, 115, 105, 32);
CMFCGDIDemoDoc* pDoc=GetDocument();
ASSERT_VALID(pDoc);
```

```
If(!pDoc)
Return;
//TODO: add draw code for native data here
}
```

NOTES

When you run this application, you will see the following output.



2.2.6 Using Color in Windows Applications

The color is one the most important that improves the visual appearance of an object. It is a non-spatial object which is added to an object. The color is used to modify the visual aspects. The Win32 API and MFC library provides several actions that you can use to take advantage of the many characteristics of colors. The RGB macro performs like a function. The RGB allows to pass three numeric values that are separated by a comma. The value of RGB must be between 0 and 255 as shown in the code given below:

```
GDIDemo View:OnDraw(CDC*pDC)
{
COLORREF color=RGB(235, 10, 220);
}
```

Another code is given below:

```
Void CMFCGDIDemo View:OnDrwa(CDC* pDC)
{
COLORREF color=RGB(235, 10, 220);
pDC->SetTextColor(color);
pDC->TextOut(95,75,L"Tutorial", 15);
CMFCGDIDemoDoc* pDoc=GetDocument();
ASSERT_VALID(pDoc);
If(!pDoc)
Return;
//TODO:add draw code for native data here
}
```

NOTES

After running the above code, the output will be as follows:

Tutorial

Pen

A pen is a tool used to draw curves and lines on a device context. It is also used to draw the borders of a geometric shape like polygon or a rectangle. There are two types of pens i.e. geometric and cosmetic. When the pen is to be assumed having different widths and various ends, then the pen is considers as geometric. A pen is considers to be cosmetic when it can be used to draw only simple lines and these lines are of fixed width. This is the feature provided by MFC having a class CPen, which encapsulates a Windows graphics device interface (GDI) pen. The code given below is related to CPen with different style values for pen.

```
Void CMFCGDIDemo View:OnDraw(CDC* pDC)
{
CPen pen;
Pen CreatePen(PS_DASHDOTDOT, 1, RGB(155, 72, 85));
pDC->SelectObject(&pen);
pDC->Rectangle(24, 34, 225, 120);
CMFCGDIDemoDoc* pDoc= GetDocument();
ASSERT_VALID(pDoc);
If(!pDoc)
Return;
//TODO: add draw code for native data here
}
```

After running the above code, the output will be as follows:

NOTES

Brush

A brush is a drawing tool used to fill out interior of lines or the closed shaped. A brush works on selecting a bucket of paint and pouring it wherever required. The class CBrush encapsulates a Windows Graphics Device Interface (GDI) brush. The code given below is related to the list of methods in CBrush class.

```
void CMFCGDIDemoView::OnDraw(CDC* pDC)
{
  CBrush brush(RGB(100, 150, 200));
  CBrush *pBrush = pDC->SelectObject(&brush);
  pDC->Rectangle(25, 35, 250, 125);
  pDC->SelectObject(pBrush);

  CMFCGDIDemoDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if (!pDoc)
     return;
  // TODO: add draw code for native data here
}
```

After running the above code, the output will be as follows:

2.3 PRINTING AND PRINT PREVIEW

Printing is one of the hardest things for accurately implementing in your Win32 program. MFC greatly simplifies this tucancode.net. You will get print preview, standardized dialogs like page setup, print setup, print job interruption dialog. The MFC source files are the better resource for help in the printing and print preview. But sometimes the MFC hides most of the functionality in members of several classes. Some steps for creating MFC printing program are:

1. Select "Single document". Make sure the check box is selected.
2. Skip step 2 and in step 3 you can disable "ActiveX Controls" since we won't use them.

3. In step 4 set the number of files to 0 and click on Advanced and delete the two bottom lines. Otherwise the program will add its document type in the right click menu in "New" and you wouldn't want this.
4. Press "Finish" and the project is ready.

There is a class CTutorialView having some member functions. The ones involved in printing are OnPreparePrinting, OnPrint, OnPrepareDC, OnBeginPrinting, OnEndPrinting, OnDraw etc. There are some steps for printing:

- To use OnDraw paint the window and OnPrint to print or paint in print preview mode.
- Right click the class CTutorialView and select "Add Virtual Function."
- Find the function **OnPrint** and press "Add and Edit".
- You are then taken to the function body and see the code given below:

```
CView::OnPrint(pDC, pInfo);
```

It is possible to make output both for the display and printer in OnDraw. This is done using pDC'!IsPrinting() and that returns TRUE if printing and FALSE if displaying. There is no requirement of OnPrint function.

After starting the program, press the Print button in the toolbar or you can use Ctrl+P also. Here, you can see the Print dialog. The Page range is set to all and the text box next to 'Pages' shows 1-65535 and this setting is default for MFC, which allows it to print only one page. You can change the page range also in OnPreparePrinting using pInfo.

The default selection and the other settings are harder. You can find all the information about member variables in the CPrintInfo. You can set the pInfo'!SetMinPage and pInfo'!SetMaxPage to set the range. For example, consider if you write the code as pInfo'!SetMaxPage(2), it means the limit of the printing job is exactly 2 pages.

2.3.1 MFC Printing Application

It is possible to know about how much pages your application requires. If it prints out something certainly small that it can be assumed as it requires 2 pages. Consider, if you print fixed-height objects like lines of text, by which you can get the size of the pages as well as the text and calculate how much pages you require. Some examples are given below for printing.

Program 1: To know exactly how much pages you require, the code is given below:

```
BOOL CTutorialView::OnPreparePrinting(CPrintInfo* pInfo)
{
pInfo->SetMaxPage(6); // or the number you need
return DoPreparePrinting(pInfo);
}
```

Program 2: Another example code where you don't know about how much pages you require before you get the user selection form the "Print" or "Print Setup" dialogs.

```
void CTutorialView::OnBeginPrinting(CDC* pDC, CPrintInfo*
pInfo)
{
int nPageHeight=pDC->GetDeviceCaps(VERTRES);
int nDocLength=GetDocument()->DocLength();
int MaxPage=max(1, (nDocLength+nPageHeight-1)/
nPageHeight);
pInfo->SetMaxPage(nMaxPage);
}
```

NOTES

Program 3: if you print the entire document contents instead printing only the current page then you have to implement the virtual function OnPrepareDC which is called before printing every page. It can be used to set the viewport, otherwise the program will print the first page every time OnPrint is called.

```
void CTutorialView::OnPrepareDC(CDC* pDC, CPrintInfo*
pInfo)
{
CView::OnPrepareDC(pDC, pInfo);
if(pDC->IsPrinting())
{
int y=(pInfo->m_nCurPage-1)*m_nPageHeight;
pDC->SetViewportOrg(0, -y);
// remove the minus sign if you are printing in MM_TEXT
}
}
```

Program 4: Following printing code will print only the appropriate lines of text.

```
for(int a=0;a<numStrings;++a)
{
Cpoint point(0,0); // start point for drawing
pDC->TextOut(str[a], point.x, point.y);
point.y-=nHeight; // if map mode is MM_TEXT change this
to +=
}
// x and y are some starting positions on x and y axis
// for MM_TEXT the point (0,0) is the top left corner of
the screen and coordinates increase to the right and down
// for MM_HIMETRIC, MM_LOMETRIC, MM_HIENGLISH and
MM_LOENGLISH
// the start is the bottom left point and coordinates
decrease(become negative) upwards and increase to the
right.
// for MM_ISOTROPIC and MM_ANISOTROPIC they are user-
defined
CPoint(x,y);
// get the current page number (for first page returns 1,
for second - 2 and so on)
nCurPage=pInfo->m_nCurPage;
```

NOTES

```
// get the index of the first CString in the array that
has to be printed in this page
nStartPos=(nCurPage-1)*linesPerPage;
// calculate the index of the last CString
nEndPos=nStartPos+linesPerPage;
// fill a TEXTMETRIC struct with various information
about the selected font
TEXTMETRIC tm;
pDC->GetTextMetrics(&tm);
int nHeight=tm.tmHeight+tm.tmExternalLeading;
for(int a=nStartPos;a<nEndPos && a<numStrings;++a)
{
pDC->TextOut(str[a], point.x, point.y);
point.y-=nHeight; // if map mode is MM_TEXT change this
to +=
}
if(a>=numStrings)
// will stop printing if all strings are printed
pInfo->m_bContinuePrinting=FALSE;
```

Program 5: An example code where you have an array of CString objects. The OnPrint code given below:

There are various methods to print the document, but first it is required to know something about mapping modes. As we know that the printers have fixed physical measures. Most of the printers support at least 600X600 dpi. That means a printer can print 600 pixels in one inch, whereas a monitor has something like 10. If you print in MM_TEXT mode, where one logical unit means one pixel the display might work for the screen. The printed images will be invisible and very small. So, the MM_ [HI/LO][ENGLISH/METRIC] map modes should be used as given below:

- MM_HIENGLISH: Each logical unit is converted to 0.001 inches
- MM_HIMETRIC: Each logical unit is converted to 0.01 millimeter
- MM_LOMETRIC: Each logical unit is converted to 0.1 millimeter
- MM_LOENGLISH: Each logical unit is converted to 0.01 inches

```
// GetDeviceCaps can give much important information about
the display device
int horzsize=pDC->GetDeviceCaps(HORZSIZE);
// gives the width of the physical display in millimeters
int vertsize=pDC->GetDeviceCaps(VERTSIZE);
// gives the height of the physical display in millimeters
int horzres=pDC->GetDeviceCaps(HORZRES);
// gives the height of the physical display in pixels
int vertres=pDC->GetDeviceCaps(VERTRES);
// gives the width of the physical display in pixels
int hdps=horzres/horzsize;
```

NOTES

```
// calculate the horizontal pixels per millimeter
int vdps=vertres/vertsize;
// calculate the verticalpixels per millimeter
// note 1: if the resolution of the printer is 600X600,
1200X1200 or anything ***X*** hdps will be equal to vdps
// note 2: multiply hdps and vdps by 2.54 to receive the
dpi
// since you didn't set the map mode it is still MM_TEXT
// now when calculating sizes in millimeters multiply
them by hdps or vdps and the sizes will be correct
CRect rectDraw=pInfo->m_rectDraw;

// this assumes the page is A$, the printer can print
without margins
// (this is not very good to assume but will work for now)
// and the page is in landscape mode (297mmx210mm)
CRect  rectOut(rectDraw. left,rectDraw.top,rect  Draw.
left+297* hdps,rect Draw.top+210*vdps);
// now print only inside this rectangle
Here, MM_TWIPS is useful while printing with text. The
method to do this is given below:
```

2.3.2 Adding Functionalities to MFC Print

If your printer supports different types of papers like A4, A3, B4, B3, Envelope, Letter, etc. But you requirement is to print on a specific type. This is done by setting the printer to defaults. However, sometimes the user require the defaults, for this, it is best to set the type of paper as well as orientation which can be portrait or landscape in the program. There should be no problem about setting anything. Here, the pInfo has a member m_pPD of type CPrintDialog* which is a pointer to the printer settings dialog. You can use it to make changes before the user starts the print job or opens the dialog.

The key is m_pPD'!m_pd which is PRINTDLG struct, which contains the hDevMode member and that is a handle to a DEVMODE data structure. It contains the information about the device initialization and environment of a printer. With the help of this pInfo'!m_pPD'!m_pd.hDevMode, you gain total control over the printing process but it is a handle you can't just set it to what you want. You have to lock the memory to it and access it instead. The handle is not set before the DoPreparePrinting function, so you can't access the data. If you access it after calling DoPreparePrinting, then the changes will take effect at the next print job. The best possible solution is to take the code of DoPreparePrinting and modify it according to the requirements.

When the user selects Print Setup from File menu, to control the defaults, it is required to associate the message with the function. You can use Ctrl+W to open ClassWizard. Associate the COMMAND message with a function (the default is OnFilePrintSetup). Select CtutorialApp for Class name and ID_FILE_PRINT_SETUP for Object ID. The code is given below:

NOTES

```
CPrintDialog pd(TRUE);
if (GetPrinterDeviceDefaults(&pd.m_pd))
{
LPDEVMODE dev = pd.GetDevMode();
GlobalUnlock(dev);
dev->dmOrientation=DMORIENT_LANDSCAPE;
dev->dmPaperSize=DMPAPER_A4;
}
DoPrintDialog(&pd);
The default function does only this:
CPrintDialog pd(TRUE);
DoPrintDialog(&pd);
```

If the user has right clicked on a document and select print from there. There is a variable cmdInfo in InitInstance, which holds the information about how is the program started. This is processed by ProcessShellCommand that you have to copy and paste in InitInstance. The ProcessShellCommand returns a Boolean value in several cases, but you don't want this to happen in InitInstance. So for this, you have to replace all the? Return?? Lines with this bResult=? Where, bResult is a Boolean variable. The code is given below:

```
case CCommandLineInfo::FilePrintTo:
case CCommandLineInfo::FilePrint:
```

Consider an example, if you want to see what and when happens in OnPreparePrinting. This is done through setting a breakpoint at the beginning of the function body after the opening curly bracket (? {?), start the program in the debugger by pressing F5.

Check Your Progress

1. What is device contexts?
2. Define the term pen class.
3. What are the two types of device objects?
4. What are the two simple help commands which are implemented by the Microsoft Foundation classes?
5. Define the term chord.

2.4 PERSISTENCE AND FILE I/O

The most important thing done by the program is to save user's data after any modification in the data. Without the capability to save edited data, the work that a user performs with an application exists only when the application is running. To create an application, while using AppWizard, Visual C++ provides much of the code that is required to save and load data.

But in some cases, while creating your own object types, you need to do some little extra work, so that to keep your user's files up to date. While writing an

NOTES

application, you have to deal with various object types, where some data objects might be simple types like integers and characters. Whereas other objects might be instances of classes, like objects created from your own custom classes or the strings from the CString class.

When you are using objects in applications that must create, save, and load documents. So, this process requires a method to save and load the state of those objects, so that you can re-create them similarly as users left them at the end of the last session. An object's capability to save and load its state is known as the persistence. Almost all MFC classes are persistent because they are derived directly or indirectly from MFC's CObject class. It provides the basic functionality for saving and loading an object's state.

The facility to write and load data from files can take a program to the next level, when writing Visual C++ programs. In order to do this, one must use the CFile class. This class enables the user to read and write data to a specific file. To begin with, one must first use a constructor. The constructor for the CFile class is straightforward and requires no parameters. To create a CFile class named file, one must simply use:

```
CFile  file
```

The CFile class is the base class for MFC file classes. This class encapsulates all the operations on the unbuffered binary input and output. The file operations include file reading, writing, positioning the file pointer before reading and writing and getting file status information.

File Handling

CFile class provides a large number of functions for opening and closing files, reading and writing the file data and performing file-oriented disk operations. All these functions have been defined in the MFC class CFile. CFile is the basic class for Microsoft foundation file classes. It directly provides unbuffered binary disk input/output services and it directly supports text files and memory files through its derivation.

CFile Class Members

Data Member

m_hFile contains the operating system file handle for an open file.

Member Functions

CFile class provides the full set of methods for manipulating binary files.

Construction methods

CFile object is created using a CFile() constructor which has three overloaded variations with zero, one or two parameters.

(1) The default constructor does not open a file , it just set s m_hFile member. The file is subsequently opened by CFlie::open() function.

```
Syntax :CFile()
```

(2) Constructs a CFile object from path of the file handle.

```
Syntax CFile(int hFile)
```

Where hFile: The handle of a file that is already open.

NOTES

(3) The third form of the constructor takes two parameters. This constructor creates the file object and opens the file with a specified access mode.

```
Syntax : CFile(LPCSTR lpszFileName, UINT nOpenFlags)
```

Where :

lpszFileName : A string defines the path of the desired file. The path can be relative or absolute.

nOpenFlags : Sharing and access modes.

File Access Mode	Purpose
CFile::modeCreate	Directs the constructor to create a new file
CFile::modeNoTruncate	Combines the value with modeCreate, if the file being created already exists, it is not truncated to length 0.
CFile::modeRead	Creates the file for reading only
CFile::modeReadWrite	Opens the file for reading and writing
CFile::modeNoInherit	File cannot be inherited by child process
CFile::modeWrite	Opens the file for writing only
CFile::shareDenyNone	Opens the file without denying any access
CFile::shareDenyRead	Opens the file and denies others read access
CFile::shareDenyWrite	Opens the file and denies others write access
CFile::shareExclusive	Opens the file and denies others all access
CFile::typeBinary	Sets binary mode
CFile::typeText	Special processing for <cr><if> pairs

Abort( )

Closes the file ignoring all warnings and errors.

```
Syntax   :   virtual void Abort( )
```

Open( )

Safely opens a file with an error-testing option and returns non–zero if the Open ( ) was successful otherwise 0.

```
Syntax :  virtual BOOL Open(LPCTSTR lpszFileName, UINT
nOpenFlags, CFileException *pError = NULL)
```

Where:

lpszFileName : A string defines path to the desired file.

nOpenflags : Same as in CFile() function.

pError : A pointer to an existing file-exception object that will receive the status of a failed operation.

Open() is designated for use with default CFile constructor.

NOTES

Close ()

Closes a file and deletes the object.

```
Syntax :  virtual void Close()
```

Duplicate ( )

Creates a duplicate file object.

```
Syntax :   virtual CFile* Duplicate( ) const
```

Input/Output function

Read()

Reads unbuffered data from a file at the current file position and returns the number of bytes transferred to the buffer.

```
Syntax :   virtual UINT Read(void * lpBuf, UINT nCount)
```

Where :

lpBuf	:	Pointer to the user-supplied buffer, that is to receive data read from the file.
nCount	:	Number of bytes to be transferred from the buffer.

Flush()

Flushes any data yet to be written.

```
Synatx : virtual void Flush()
```

Write ( )

Writes unbuffered data to a file at the current file position and returns the number of bytes transferred from the buffer.

```
Sybtax : virtual void Write( const void* lpBuf, UINT
nCount )
```

Where :

lpBuf	:	Pointer to the user-supplied buffer, from where data is to be taken for writing.
nCount	:	Number of bytes to be transferred to the buffer.

File position functions

CFile provides many file pointer positioning functions. The file pointer is positioned for next reading and writing operations.

Seek()

Positions the current file pointer at the desired location.

```
Syntax : virtual LONG Seek(LONG IOff, UINT nFrom)
```

Where :

IOff	:	Number of bytes to move the pointer
nFrom	:	Pointer movement mode
CFile:: Begin	:	Moves the file pointer IOff bytes forward from the begining of the file.
CFile ::Current	:	From the current position in the file.
CFile:: End	:	Backward from the end of the file.

SeekToBegin()

Positions the current file pointer at the beginning of the file.

```
Syntax :    void SekkToBegin()
```

SeekToEnd()

Positions the current file pointer to the end of the file.

```
Syntax :  void SeekToEnd()
```

GetLength()

Retrieves the logical length of the file.

```
Syntax :  virtual DWORD GetLength() const
```

SetLength ( )

Changes the length of the file.

```
Sybtax :  virtual void SetLength( ULONGLONG dwNewLen);
```

Remove ()

Deletes the specified file.

```
Syntax : virtual DWORD Remove()
```

File locking functions

File locking functions prevent the data in the disk file. Cfile provides two file locking functions.

LockRange ( )

This function locks the range of the bytes in a file.

```
Sybtax : virtual void LockRange(ULONGLONG dwPos,  ULONGLONG
dwCount  );
```

Where:

DwPos : Byte offset of the start of the range to be locked.

DwCount : Number of bytes to be locked.

UnLockRange ()

Unlocks a range of bytes in a file.

```
Syntax  :virtual  void  UnlockRange(  ULONGLONG  dwPos,
ULONGLONG dwCount );
```

Where:

DwPos : Byte offset of the start of the range to be unlocked.

DwCount : Number of bytes to be unlocked.

File status functions

Cfile provides many functions to report the status of the file.

GetFileName ()

Returns filename of the selcted file.

```
Syntax : virtual CString GetFileName( ) const
```

GetFIle Title ()

Returns title of the selected file.

```
Syntax : virtual CString GetFileTitle( ) const
```

GetFilePath ()

Returns path of the selected file

```
Syntax :  virtual CString GetFilePath( ) const
```

GetPosition ()

Returns position the file pointer.

```
 Syntax :virtual ULONGLONG GetPosition(
```

Example:

```
class CMainFrame : public CMDIFrameWnd
{
private:
    CFile fp;
    struct record
    {
       char name[20];
       int age;
       float salary;
    }

public:
    CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
       // NOTE - the ClassWizard will add and remove
member functions here.
```

NOTES

```
        //    DO NOT EDIT what you see in these blocks of
generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here

Create(0, "Writing and Reading Data", WSOVERLAPPEDWINDOW,
rectDefault, 0, MAKEINTRESOURCE(IDR_MENU1));
    fp.Open("test.txt", CFile::modeCreate |
CFile::modeReadWrite);

}

void CMainFrame::writedata()
{
int i;
record e[] = {
              {"AAA", 11,1111,11f};
              {"BBB", 22,2222,22f}
              {"CCC", 33,3333,33f}
              {"DDD", 44,4444,44f}
           }

fp.SeekToBegin();
for(i=0 ;i<3;i++)
fp.Write(&e[i], sizeof(record));

}

void CMainFrame::readdata()
{
    record temp;
    char str[100];
    if(fp.GetLength() ==0)
       MessageBox("File is Empty", "ReadRecord.");
    fp.SeekToBegin();
    while(fp.Read(&temp, sizeof(record) !=0)
    {
       sprintf(str, "Name: %s \n AGe: %d \n Salary : %2f"
, temp.name , temp.age , temp.salary);
```

```
        MessageBox(str, "Record...");
    }

}

void CMainFrame::OnDestroy()
{
    fp.Close();
    CMDIFrameWnd::OnDestroy();

    // TODO: Add your message handler code here

}

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_COMMAND(101,writedata);
    ON_COMMAND(201,readaata);
    ON_WM_CREATE()
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

NOTES

The Menu (IDR_MENU1) contains two menu items with IDs 101 and 201 and two message handling functions `writedata()` and `readdata()` corresponding to these menu items respectively.

CFile Derivation CStdioFile

CStdioFile object provides a CFile interface to buffer the stream disk files usually in the text mode. A CStdioFile object represents a C run-time stream file as opened by the run-time function open. Stream files are buffered and can be opened in either the text or binary mode.

Class Member

The class member comprises the following:

Data member

mpStream: Contains a pointer to open file.

Member functions

CstdioFile provides two additional functions of Text I/O operations.

CStdioFile()

Constructs a CStdioFile object from a path or file pointer.

```
Syntax : CStdioFile(LPCTSTR lpszFileName, UINT
nOpenFlags)
```

lpszFileName : Specifies the pointer to the user-supplied buffer that will receive a null-terminated text string.

nOpenFlags : Shares an access mode, specifies the action to take when the file is opened.

ReadString()

Reads a single line of the text and returns the pointer to the buffer containing the text data, NULL if end of file was reached.

```
Syntax : virtual LPTSTR ReadString(LPTSTR lpsz, UINT nMax)
```

lpsz : Specifies pointer to the user-supplied buffer that will receive a null-terminated text string.

nMax : Specifies the maximum number of characters to read, it should be one less than the size of the lpsz buffer.

WriteString()

Writes a single line text.

```
Syntax : virtual void WriteString(LPCTSTR lpsz);
```

Lpsz : Specifies a pointer to a buffer containing a null-terminated text String..

Example:

```
class CMainFrame : public CMDIFrameWnd
{
private:
    CStdioFile fp;

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here

Create(0, "Writing and Reading Data",
WSOVERLAPPEDWINDOW, rectDefault, 0,
MAKEINTRESOURCE(IDR_MENU1));
    fp.Open("test.txt", CFile::modeCreate |
CFile::modeReadWrite);


}

void CMainFrame::writedata()
{
c str[40];
int i;
struct  record
{
    char name[2];
```

NOTES

```
    int age;
    float salary;
}
record e[] = {
              {“AAA”, 11,1111,11f};
              {“BBB”, 22,2222,22f}
              {“CCC”, 33,3333,33f}
              {“DDD”, 44,4444,44f}
           }

fp.SeekToBegin();
for(i=0 ;i<3;i++)
{
sprintf(str, “%s %d %2f\n”, e[i].name, e[i].age,
e[i].salary);
fp/WriteString(str);
}
}

void CMainFrame::readdata()
{
    record temp;
    char str[100];
    if(fp.GetLength() ==0)
       MessageBox(“File is Empty”, “ReadRecord.”);
    fp.SeekToBegin();
    while(fp.Read(&temp, sizeof(record) !=0)
    {
       sprintf(str, “Name: %s \n AGe: %d \n Salary :
%2f” , temp.name , temp.age , temp.salary);
       MessageBox(str, “Record...”);
    }

}

void CMainFrame::readdata()
{

    CString str;
    char nm[20], temp[20];
    int ag;
    float sr;
       if(fp.GetLength() ==0)
       MessageBox(“File is Empty”, “ReadRecord.”);
```

NOTES

```
    fp.SeekToBegin();
    while(fp.ReadString(str) != NULL)
    {
       Sscanf(str, "%s %d %f",nm,ag,sr);
       str = "Name:";
       str += nm;
       str += \nAge:";
          sprintf(temp, "%d",ag);
       str += temp;
       str += "\nSalary:";
       sprintf(temp, "%2f", sr);
       str += temp;
       MessageBox(str, "Record...");
    }

}

void CMainFrame::OnDestroy()
{
    fp.Close();
    CMDIFrameWnd::OnDestroy();

    // TODO: Add your message handler code here

}

    DECLARE_MESSAGE_MAP()
};
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_COMMAND(101,writedata);
    ON_COMMAND(201,readaata);
    ON_WM_CREATE()
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Menu (IDR_MENU1) contains two menu items with IDs 101 and 201 and two message handling functions `writedata()` and `readdata()` corresponding to these menu items respectively.

CMemFile Class

The CMemFile class object represents a file that completely resides in RAM. The memory file does not have any corresponding file in the disk. The memory file is a piece of memory used to transfer the data between two running processes. It

serves the purpose of an inter process communication. Memory files can be accessed through the Cfile member functions.

CMemFile Class Members

NOTES

The CmemFile( ) constructor function creates a CmemFile object and opens a memory file. It automatically allocates it a memory. Alternatively, a memory block can be attached to the CmemFile object using the `CmemFile:: Attach()` function.

Cfile :: h_mFile data member has no meaning in case of the CmemFile class. The `Duplicate()`, `LockRange()` and `UnLockRange()` functions of the Cfile class can be used with the CMemFile object.

Alloca ()

This function is used to modify the memory allocation block . This virtual function is overridden in the program. It returns the pointer to the allocated memory block.

```
Synrax : virtual BYTE* Alloc(SIZE_T nBytes );
```

where :

nBytes : Number of bytes to be allocated.

Attach ()

Attaches a memory block to the CMemFile object.

```
Syntax  void Attach(BYTE* lpBuffer, UINT nBufferSize,
UINT nGrowBytes);
```

Where:

LpBuffer	:	The pointer to buffer , attach to CMemFile object.
NBufferSize	:	The size of the buffer in an integer number of bytes.
NGrowBytes	:	The number of bytes to be incremented.

Detach()

Detaches a memory block from the CMemFile object. It returns the pointer to the memory block containing the CMemFile object. This function also closes the CMemFile file.

```
Syntax : BYTE * Detach( );
```

Free ()

A virtual function that can be overridden to deal with the located memory block containing the CmemFile object.

```
Syntax : virtual void Free(   BYTE * lpMem );
```

Where :

lpMem : The pointer to the memory block that has to be freed.

GrowFile ()

This function is called to increase the size of the CMemFile object. Originally this calls the `Realloc ( )` function . This function needs to be overridden.

```
Syntax : virtual void GrowFile(   SIZE_T dwNewLen );
```

Where : dwNewLen : New size of the memory file.

MemCopy ( )

This virtual function is overridden to copy the data from or to the memory file. This funtion is called the program override `Read()` and `Write()` functions of the Cfile object .

```
Syntax : virtual BYTE* Memcpy(  BYTE* lpMemTarget,  const
BYTE* lpMemSource,
  SIZE_T nBytes);
```

Where :

lpMemTarget	:	Pointer to the target memory block where data to be copied.
lpMemSource	:	Pointer to the source memory block from where data to be copied.
nBytes	:	Number of bytes to be copied.

`Realloc()`

A virtual function that is overridden to reallocate memory block . It returns pointer to the reallocated block.

```
Syntax : virtual BYTE* Realloc(  BYTE* lpMem,   SIZE_T
nBytes );
```

Where :

lpMem	:	Pointer to memory block that is to reallocate.
nBytes	:	New size of the reallocated block.

2.4.1 Basic File Operations

The CFile class basically handles the file accessing and manipulating operations, such as opening, closing, reading and writing, but it does not deal with the file copying, moving, renaming and file deleting operations. The Win 32 API functions are used to do such operations in an MFC program.

Copying a File

The Win 32 API function `CopyFile()` is used to copy an existing file to a new file. On success this function returns to nonzero, on failure it returns to zero.

```
Syntax : BOOL WINAPI CopyFile(LPCTSTR
lpExistingFileName, LPCTSTR lpNewFileName, BOOL
bFailIfExists);
```

Where:

`lpExistingFileName`	:	`Name the existing file.`
`lpNewFileName`	:	`Name of the new file.`
`bFailIfExists`	:	`If this parameter is TRUE and the new file is already existing, the CopyFile ( ) function fails. If it is FALSE, the new file overwritten.`

NOTES

Moving a File

The Win 32 API function `MoveFile()` is used to move an existing file or directory with its subdirectories, to a new file. On success this function returns nonzero, on failure it returns to zero.

```
Syntax : BOOL WINAPI MoveFile( LPCTSTR lpExistingFileName,
LPCTSTR lpNewFileName );
```

Where:

LpExistingFileName	:	The current name of the file or directory on the local system.
LpNewFileName	:	The name of the new file or directory. The new file can be on the different file system or drive but the directory should be on the same drive.

Renaming a File

The Win 32 `MoveFile()` or `MoveFileEx()` functions are also used to rename a file. To rename a file the second parameter is given a new name.

Deleting a File

The Win 32 API `DeleteFile()` function is used to delete a file. On success this function returns to nonzero, on failure it returns to zero.

```
Syntax : BOOL WINAPI DeleteFile(LPCTSTR lpFileName);
```

Where :

lpFileName : Name of the file to be deleted.

2.4.2 Files and Windows Applications

When you are using the Visual C++'s AppWizard to create a program, then you get an application that uses view classes and document to organize, edit, and display its data. As we know that the document object is derived from the CDocument class which is responsible for holding the application's data during a session. It is also used for saving and loading the data so that the document persists from one session to another. Here, you will use a File Demo application which shows about the basic techniques behind saving and loading data of an object derived from CDocument. File Demo's document is a single string comprising a short message, which the view displays. Three menu items are related in the File Demo application.

When the program begins, the message is automatically set to the string Default Message. The users can change this message. The file, save menu option is used to save the document, and the file, open, reloads it from the disk.

While working with an application created using AppWizard, there is requirement to complete a number of steps to enable your document to save and load its state. The steps are as follows:

1. First you have to define the member variables that will hold the document's data.

NOTES

2. Initialize the member variables in the document class's OnNewDocument() member function.

3. You have to display the current document in the view class's OnDraw() member function.

4. Provide member functions in the view class that enable users to edit the document.

5. Add to the document class's Serialize() member function the code requires to save and load the data that comprises the document.

When the application can handle multiple documents, then it is required to do a little extra work and to be sure that you can use, change, or save the correct document.

If you want to build the File Demo application start by using AppWizard to create an SDI application. All the other AppWizard choices should be left at their default values. This is done because it will speed things up by clicking Finish on Step 1 after selecting SDI.

Double-click CfileDemoDoc in ClassView to edit the header file for the document class. In the Attributes section, you can add a CString member variable known as m_message. The attributes section is given below:

```
// Attributes
public:
    CString m_message;
```

Here, the document's storage is more than a single string object. Typically, the document's storage requirements are more complex. The single string is sufficient to demonstrate the basics of a persistent document. The MFC programmers mostly use public variables in their documents, instead a private variable with public access functions which makes it simpler to write the code in the view class and that will access the document variables. You have to expand CFileDemoDoc in ClassView and double-click OnNewDocument() to edit it. To initialize the string, add a line of code as shown below that shows about initializing the document's data.

```
BOOL CFileDemoDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
    return FALSE;
    m_message = "Default Message";
    return TRUE;
}
```

With the help of document class's m_message data member initialized, the application can show the data in the view window. For this, you have to edit the view class's OnDraw() function as shown in the code given below. Expand CFileDemoView in ClassView and double-click OnDraw() to edit it. Given below is the example that represents about the displaying the document's data:

```
void CFileDemoView::OnDraw(CDC* pDC)
{
    CFileDoc* pDoc = GetDocument();
```

```
    ASSERT_VALID(pDoc);
    pDC->TextOut(20, 20, pDoc->m_message);
}
```

NOTES

Now, build File Demo to make sure that there are no typos, and run it. You should see Default Message appears onscreen. You need to allow users to edit the application's document by changing the string. The application should display a dialog box which helps the user to enter any particular string. So, you have the edit, change message menu option to assign the string a different, hard-coded value.

Click the Resource tab to expand the resources, switch to ResourceView, to expand menus. You have to double-click IDR_MAINFRAME to edit it. By clicking on the edit item in the menu, you are editing to drop it down. At the end of the list, click the blank item and type change & message, so this will add another item to the menu. Select view, ClassWizard to make the connection between the code and this menu item.

There is ID_EDIT_CHANGEMESSAGE highlighted and if it is not displayed, then click it in the box on the left to highlight it. From the drop-down box on the upper right, you can select CFileDemoView. Click COMMAND in the lower-right box and then click the Add Function button. Accept the suggested name, OnEditChangemessage(), by clicking OK on the dialog that appears. Click the edit code to open the new function in the editor, write the code as shown below, that represents about changing the document's data.

```
void CFileDemoView::OnEditChangemessage()
{
  CTime now = CTime::GetCurrentTime();
   CString changetime = now.Format("Changed at %B %d
%H:%M:%S");
  GetDocument()->m_message = changetime;
  GetDocument()->SetModifiedFlag();
  Invalidate();
}
```

This function responds to the application's change message command, edit, builds a string from the current date and time and transfers it to the document's data member. The call to the document class's SetModifiedFlag() function informs about the object that its contents have been changed. The application will inform about exiting with unsaved changes as long as you remember to call SetModifiedFlag() and all over there might be a change to the data. Finally, this code forces a redraw of the screen by calling Invalidate ().

2.4.3 Serialization

Consider an example of the document class serialize() function. The code is given below:

```
void CFileDoc::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
```

NOTES

```
    {
        // TODO: add storing code here
}
    else
    {
        // TODO: add loading code here
}
}
```

The CString class defines the >> and << operators for transferring strings to and from an archive. It is a simple task to save and load the document class's data. The code is given below and you have to simply add this line:

```
ar << m_message;
```

Add this similar line where the loading code belongs:

```
ar >> m_message;
```

The >> operator fills m_message from the archive. The << operator sends the CString m_message to the archive. All the document's member variables are simple data types like integers or characters, or MFC classes like CString with these operators. It is easy to save and load the data. The operators that are defined for these simple data types are int, BYTE, WORD, DWORD, LONG, float and double.

Build the file demo and run it. Select change message, edit and finally check the new string onscreen. You have to select file, save and then enter a filename. Now select file, new and then save the current changes. After that, select file, ppen and browse to your file and re-open it. Finally, you can see the file demo that may be saved and reload a string.

Consider an example, where you require to improve the File Demo application so that it consists of its data in a custom class called CMessages. M_messages are known as the member variables and CMessages is the instance of that. Three CString objects are held by this class and each of which must be saved and loaded for the application so that it will work correctly. The code is given below regarding the one way to save the new class's strings.

```
void CFileDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        ar << m_messages.m_message1;
        ar << m_messages.m_message2;
        ar << m_messages.m_message3;
    }
    else
    {
        ar >> m_messages.m_message1;
        ar >> m_messages.m_message2;
        ar >> m_messages.m_message3;
```

```
}
}
```

NOTES

In the above code, there are three member variables of the CMessages class which are public. If the class is changed in any way, then this code also has to be changed. It is more object oriented to delegate the work of storing and loading to the CMessages class itself which needs some preparation. Some basic steps used to create a class that can serialize its member variables are given below.

1. First derive the class from CObject.
2. Secondly, place the DECLARE_SERIAL () macro in the class declaration.
3. Place the IMPLEMENT_SERIAL () macro in the class implementation.
4. You have to override the Serialize () function in the class.
5. Provide an empty, default constructor for the class.

The code given below is to build an application that creates persistent objects. Following are the steps for doing this.

- Consider sample application, File Demo 2, which shows the steps you take to create a class from which you can create persistent objects.
- It will have an Edit, Change Messages command that changes all three strings.
- Like File Demo, it will save and reload the document when the user chooses File, Save or File, Open.
- Build an SDI application called MultiString just as you built File Demo.
- Add a member variable to the document, as before, so that the Attributes section of MultiStringDoc.h reads.

```
// Attributes
public:
    CMessages m_messages;
```

The next step is to write the CMessages class. Now, check the CMessages Class. You have to check how the CMessages class, of which the document class's m_messages data member in an object, works. While doing work with this class, you can check that how to implement the preceding five steps for creating a persistent class.

First choose Insert, New Class to create the CMessages class. Change the class type to generic class and give it a name like CMessages. At the bottom of the screen, enter CObject as the base class name and leave the column set to public.

There are two files that has been created: messages.cpp for the code and the messages.h for the header. It also adds some very simple code to these files for you. Go to the Multistringdoc.h and add this line before the class definition:

```
#include "Messages.h"
```

This will confirm that the compiler knows about the CMessages class when it compiles the document class. If you want to be sure you haven't forgotten anything, you can build the project now. Now, go to the Messages.h and add the code given below:

NOTES

```
DECLARE_SERIAL(CMessages)
protected:
    CString m_message1;
    CString m_message2;
    CString m_message3;
public:
    void SetMessage(UINT msgNum, CString msg);
    CString GetMessage(UINT msgNum);
    void Serialize(CArchive& ar);
```

Here, the DECLARE_SERIAL () macro offers the member variable declarations and additional function that are required to implement object persistence. The class's data members are three objects of the CString class and are the protected member variables. GetMessage() is the complementary function that enables a program to retrieve the current value of any of the strings.

SetMessage(), whose arguments are the index of the string to set and the string's new value, changes a data member. GetMessage() is the complementary function, enabling a program to retrieve the current value of any of the strings. Its single argument is the number of the string to retrieve. The class overrides the Serialize () function where all the data loading and saving takes place. The Serialize () function is the heart of a persistent object, with each persistent class implementing it in a different way. The code given below of MESSAGE.CPP shows the CMessages class implementation file.

```
void CMessages::SetMessage(UINT msgNum, CString msg)
{
     switch (msgNum)
      {
     case 1:
          m_message1 = msg;
          break;
     case 2:
        m_message2 = msg;
          break;
     case 3:
        m_message3 = msg;
          break;
      }
   SetModifiedFlag();
}
CString CMessages::GetMessage(UINT msgNum)
{
   switch (msgNum)
    {
      case 1:
         return m_message1;
```

NOTES

```
        case 2:
            return m_message2;
        case 3:
            return m_message3;
        default:
            return "";
    }
}
void CMessages::Serialize(CArchive& ar)
{
    CObject::Serialize(ar);
    if (ar.IsStoring())
    {
        ar << m_message1 << m_message2 << m_message3;
    }
    else
    {
        ar >> m_message1 >> m_message2 >> m_message3;
    }
}
```

Here, the SetMessage() and GetMessage() functions perform their assigned tasks accurately. The Serialize () function may inspire a couple of questions. The first line of the body of the function calls the base class's Serialize () function. This is a typical practice for many functions that override functions of a base class.

Here, the call to CObject::Serialize () doesn't do much because the CObject class's Serialize () function is empty. Calling the base class's Serialize () function is a good practice to get into this because you can't always be working with classes derived directly from CObject.

After calling the base class's version of the function, Serialize () saves and loads its data in the same way a document object does. The data members that must be serialized are CString objects. The program can use the >> and << operators to write the strings to the disk. On the top of messages.cpp, after the include statements, add the code statement given below.

```
IMPLEMENT_SERIAL(CMessages, CObject, 0)
```

Here, the IMPLEMENT_SERIAL () macro is significant partner to the DECLARE_SERIAL () macro. It provides the implementation for the functions that give the class its persistent competencies. The macro's three arguments are the name of the class, a schema number and the name of the immediate base class.

CMessages is defined and implemented so that the member functions of the MultiString document and view classes can work with it. First, you have to expand CMultiStringDoc and double-click OnNewDocument() to edit it. Add the following code in place of the TODO comments.

```
m_messages.SetMessage(1, "Default Message 1");
m_messages.SetMessage(2, "Default Message 2");
m_messages.SetMessage(3, "Default Message 3");
```

The document class initializes each string by calling the CMessages class's SetMessage() member function because it can't directly access the data object's protected data members. You have to expand CMultiStringView and double-click OnDraw() to edit it. You have to write the code given below.

```
void CMultiStringView::OnDraw(CDC* pDC)
{
       CMultiStringDoc* pDoc = GetDocument();
       ASSERT_VALID(pDoc);
   pDC'!TextOut(20, 20, pDoc'!m_messages.GetMessage(1));
   pDC'!TextOut(20, 40, pDoc'!m_messages.GetMessage(2));
   pDC'!TextOut(20, 60, pDoc'!m_messages.GetMessage(3));
}
void CMultiStringView::OnEditChangemessages()
{
  CMultiStringDoc* pDoc = GetDocument();
  CTime now = CTime::GetCurrentTime();
   CString changetime = now.Format("Changed at %B %d
%H:%M:%S");
  pDoc->m_messages.SetMessage(1, CString("String 1 ") +
changetime);
  pDoc->m_messages.SetMessage(2, CString("String 2 ") +
changetime);
  pDoc->m_messages.SetMessage(3, CString("String 3 ") +
changetime);
  pDoc->SetModifiedFlag();
  Invalidate();
}
```

Similarly for the file demo, you can add a "Change Messages" item to the Edit menu. You can connect it to a view function called OnEditChangemessages. This function will change the data by calling the CMessages object's member functions, as shown in the following code. The view class's OnDraw() function also calls the GetMessage() member function to access the CMessages class's strings. The code for editing the data strings is given below.

All that remains is to write the document class's Serialize () function, where the m_messages data object is serialized out to disk. You just give the work to the data object's own Serialize() function, as shown in the following code, that represents serializing the data object.

```
void CMultiStringDoc::Serialize(CArchive& ar)
{
   m_messages.Serialize(ar);
        if (ar.IsStoring())
         {
         }
        else
         {
```

```
        }
}
```

NOTES

After serializing the m_messages data object, nothing much is left to do in the document class's Serialize() function. The call to m_messages.Serialize() passes the archive object as its single parameter. Now build MultiString and test it in the same way you tested File Demo.

Check Your Progress

6. What is the function of the CFile class?
7. Define a CFile() constructor.
8. What is the function of a CFile Derivation CStdioFile?
9. What does the nOpenFlags do?
10. What does the CMemFile Class object represent?
11. What is the function of a CMemFile Constructor?
12. Which functions does the work of copying, moving, renaming and deleting of files?

2.5 ANSWER TO 'CHECK YOUR PROGRESS'

1. The device context consists of the information about the application, the system and the window in which you are drawing any type of graphics.
2. In the pen class, CPen, is used to identify the width and color width for drawing lines on the screen. It is the primary resource tool for drawing any type of line on the screen.
3. There are two types of device objects that are: CClientDC and CWindowDC.
4. There are two simple help commands which are implemented by the Microsoft Foundation Classes:
   - ID_HELP_USING which is implemented by CWinApp::OnHelpUsing
   - ID_HELP_INDEX which is implemented by CWinApp::OnHelpIndex
5. A chord is an arc which is having two ends and these are connected by a straight line.
6. The CFile is the basic class for Microsoft foundation file classes. It directly provides unbuffered binary input/output services and it directly supports the text files and the memory files through its derivation.
7. A CFile constructor creates a CFile object and has three overloaded variations with zero, one or two parameters.
8. A CFile Derivation CStdioFile provides a CFile interface to buffer the stream disk files usually in the text mode.
9. The nOpenFlags member function shares an access mode and specifies the action to take when a file is opened.

10. The CMemfile class object represents a file that completely resides in RAM.
11. The CMemFile() constructor function creates a CMemFile object and opens a memory file.
12. The Win 32 API functions are used to do operations like copying, moving, renaming and deleting of files.

2.6 SUMMARY

- The Windows OS provides several levels of abstraction. These levels are used for creating and drawing by using graphics in the applications. In the past, while using DOS programming, there is requirement to exercise a better control over the graphics hardware, which is used to draw any type of image in an application.
- The device context consists of the information about the application, the system and the Window in which you are drawing any type of graphics.
- A Windows GDI object type is denoted by an MFC library class. For the GDI object classes, the *CGdiObject* is the abstract base class.
- A Windows GDI object is signified by a C++ object of a class derived from *CGdiObject*.
- Device-independent drawing in Windows is allowed by the Device contexts.
- A Polyline is a series of connected lines. In this the lines are stored in an array of CPoint or POINTvalues.
- A chord is an arc which is having two ends and these are connected by a straight line. The arcs we have drawn so far are considered open figures because they are made of a line that has a beginning and an end (unlike a circle or a rectangle that do not).
- A pen is a tool that used to draw curves and lines on a device context. A pen is also used to draw the borders of a geometric shape like polygon or a rectangle.
- The CFile class is the base class for MFC file classes.
- The CFile class encapsulates all the operations on the unbuffered binary input and output. It provides functions for opening and closing files, reading and writing the data and performing file-oriented disk operations.
- The CFile object is created using a CFile ( ) constructor which has three overloaded variations with zero, one or two parameters.
- The CFile provides many file pointer positioning functions. The file pointer is positioned for the next reading and writing operations.
- The CFile also provides the file locking functions which prevent the data in the disk file.
- A CStdioFile object represents a C run-time stream file as opened by the run-time function fopen. The stream files are buffered and can be opened in either the text mode or the binary mode.

- The memory file is a piece of memory used to transfer the data between the two running processes. It serves the purpose of an interprocess communication.

NOTES

2.7 KEY TERMS

- **Device Contexts:** The device context consists of the information about the application, the system and the Window in which you are drawing any type of graphics.
- **CFont**: It is a font which is complete collection of characters of specific type and specific size. Fonts are mostly stored on disk like device-specific or as resources.
- **CBrush:** It is used to describe a bitmapped pattern of pixels, which is used to fill the areas with the help of colors.
- **CFile Class:** It encapsulates all the operations on the unbuffered binary input and output.
- **Construction Methods:** It creates a CFile object using a CFile constructor which has three overloaded variations with zero, one or two parameters.
- **Read():** It reads the unbuffered data from a file at the current file position and returns the number of bytes transferred to the buffer.
- **Seek():** It positions the current file pointer at the desired location.
- **CMemFile Class Object:** It represents a file that completely resides in RAM.
- **Win 32 API Function MoveFile ( ):** It is used to move an existing file or directory with its subdirectories, to a new file.

2.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is Device context
2. How will you find print preview in program documents?
3. How will you define the persistence and file I/O?
4. What is the difference between disk file and memory file?
5. Which operations are not supported by the CFile class?
6. Which function is used to find the length of the file?
7. How many types of methods/functions are supported by CFile class?

Long-Answer Questions

1. Discuss briefly about the drawing on the screen with the help of examples.
2. Explain the printing and print preview of program documents.

3. Elaborate on the persistence and file I/O with the help of relevant examples.
4. Write an MFC program to convert all lower case characters to the upper case.
5. Write an MFC program to add the salary records of ten employees (basic, HRA,CCA etc.), to find the total salary of each employee and also find the average salary.
6. Write an MFC program to add, modify and delete records in a Database file.
7. Explain in detail the file positioning and file locking functions of the CFile.
8. Discuss the various file operations.

2.9 FURTHER READING

Cornell, Gary. 1998. Visual Basic 6 from the Ground Up. New Delhi: Tata McGraw-Hill.

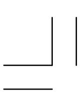
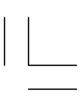
Manchanda, Mahesh. 2009. Visual Programming. New Delhi: Vikas Publishing House Pvt. Ltd.

Balena, Francesco. 1999. Programming Microsoft Visual Basic 6.0. Bangalore: WP Publishers and Distributors (P) Ltd.

Petroutsos, Evangelos. 1998. Mastering Visual Basic 6, 1st Edition. New Delhi: BPB Publications.

Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.

Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 3 STATUS BARS, TOOL BARS, COMMON CONTROLS, HELP, PROPERTY PAGES AND SHEETS

Structure

3.0 Introduction
3.1 Objectives
3.2 Status Bars and Tool Bars
3.3 Common Controls
  3.3.1 Check Box Control
  3.3.2 Radio Button Control
  3.3.3 List Box Control
  3.3.4 Combo Box Control
  3.3.5 Slider Control
3.4 Building Blocks of Help
3.5 Property Pages and Sheets
3.6 Answer to 'Check Your Progress'
3.7 Summary
3.8 Key Terms
3.9 Self-Assessment Questions and Exercises
3.10 Further Reading

3.0 INTRODUCTION

A status bar is a graphical control element which poses an information area typically found at the window's bottom. It can be divided into sections to group information. Its job is primarily to display information about the current state of its window, although some status bars have extra functionality. The toolbar, also called a bar or standard toolbar (originally known as ribbon) is a graphical control element on which on-screen icons can be used. A toolbar often allows for quick access to functions that are commonly in the program. The Command Button adds a clickable button to the form that may be used to run a specified Check Code block. Here are a few examples of how a Command Button and the Check Code behind the button might be used: Field values are compared, and automatic calculations are performed. There are some helper functions() which are building blocks of Help. When you call the SetSize() function member of an array collection, a global function, ConstructElements(), is called. This function is used to allocate memory for the number of elements that you want to store in the array collection.

A property sheet is a dialog box that consists of property pages. It is also known as a tab dialog box. It is enclosed on a page with the help of a tab. To select a set of controls, you can click a tab in the property sheet. It is based on a dialog template resource and consists of various controls.

In this unit, you will learn about the status bars and tool bars, common controls, building blocks of help, property pages and sheets.

NOTES

3.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the significance of status bars and tool bars
- Analyse the common controls
- Discuss about the building blocks of help
- Know about the property pages and sheets

3.2 STATUS BARS AND TOOL BARS

Windows makes the computers easier to use and learn and this was the driving objective behind the development of Graphical User Interfaces (GUI). The GUI designers states that a standard set of menus should be used for all applications and these menus should be organized in a uniform manner. The application designers found that new users still had difficulty in learning new applications. This is the reason that the application designers designed toolbars as one solution to both problems.

A toolbar is a small band which is attached to the dialog window or a window frame. It is floating independent of the application frame. This dialog has small number of buttons consisting of graphic images and these images could be used in place of the menus. Designers place the most frequently used functions for their applications on these toolbars. The information bar which was placed at the bottom of application windows and that provides detailed descriptions of toolbar buttons and menu entries.

You can add a few additional toolbar buttons to the default toolbar that the AppWizard creates when you start a new SDI or MDI application. In the Visual C++ designer, you can pull up the toolbar with the help of the resource view in the workspace pane and start adding new buttons. Like as in the menu designer, the end of the toolbar at all times has a blank entry and waiting to turn it into another toolbar button, as shown in Figure 3.1.

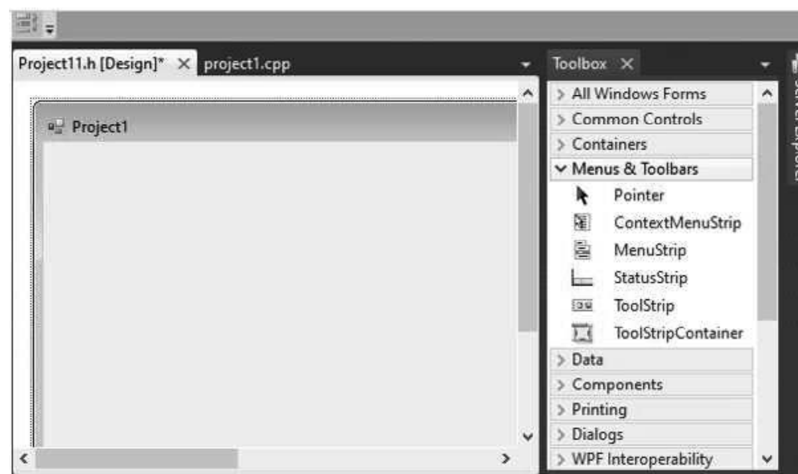


Fig. 3.1 Toolbar Designer

In the previous SDI and MDI applications, there is no additional functionality that required you to touch the frame window. You have to start adding and modifying code in that module because the toolbar is attached to the frame. When you open the CMainFrame class to the OnCreate function, it can be seen that it is creating the existing toolbar and then the toolbar is being attached to the frame in this function.

Before adding the toolbar to the application frame, it is required to add a variable to the CMainFrame class to hold the new toolbar. To add the color toolbar to your drawn application, you have to right-click the CMainFrame class in the class view tab of the workspace pane. From the pop-up menu, Select add member variable and identify the variable type as CToolBar, the name as m_wndColorBar and access as protected.

After adding the variable to your toolbar, it is required to add some code in the OnCreate function in the CMainFrame class to add the toolbar and attach it to the frame. The code given below is to make the modifications to add the color toolbar to your drawing application. This code represents the modified CMainFrame.OnCreate function.

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
return -1;
if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
WS_CHILD | WS_VISIBLE | CBRS_TOP
 | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_SIZE_DYNAMIC) ||
 !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
 {
 TRACE0("Failed to create toolbar\n");
 return -1; // fail to create
 }
 //////////////////////
 // MY CODE STARTS HERE
 //////////////////////
// Add the color toolbar
 int iTBCtlID;
 int i;
// Create the Color Toolbar
if (!m_wndColorBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD
|
WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
 !m_wndColorBar.LoadToolBar(IDR_TBCOLOR))
{
TRACE0("Failed to create toolbar\n");
```

NOTES

```
return -1; // fail to create
}
 // Find the Black button on the toolbar
iTBCtlID = m_wndColorBar.CommandToIndex(ID_COLOR_BLACK);
if (iTBCtlID >= 0)
{
 // Loop through the buttons, setting them to act as
radio
buttons
for (i= iTBCtlID; i < (iTBCtlID + 8); i++)
m_wndColorBar.SetButtonStyle(i, TBBS_CHECKGROUP);
 }
 //////////////////////
 // MY CODE ENDS HERE
 //////////////////////
if (!m_wndStatusBar.Create(this) ||
!m_wndStatusBar.SetIndicators(indicators,
sizeof(indicators)/sizeof(UINT)))
{
TRACE0("Failed to create status bar\n");
return -1; // fail to create
}
// TODO: Delete these three lines if you don't want the
toolbar to
// be dockable
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
//////////////////////
// MY CODE STARTS HERE
 //////////////////////
 // Enable docking for the Color Toolbar
 m_wndColorBar.EnableDocking(CBRS_ALIGN_ANY);
 //////////////////////
 // MY CODE ENDS HERE
 //////////////////////
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar);
//////////////////////
// MY CODE STARTS HERE
//////////////////////
// Dock the Color Toolbar
DockControlBar(&m_wndColorBar); 76:
 //////////////////////
// MY CODE ENDS HERE
//////////////////////
```

```
return 0;
}
```

Creating the Toolbar

NOTES

Following code is the first part that you added to create toolbar.

```
if (!m_wndColorBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD
|
WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
!m_wndColorBar.LoadToolBar(IDR_TBCOLOR))
```

It consists of two distinct functions, which are required for creating a toolbar. Here, the first function is CreateEx that creates the toolbar itself. And the second function is LoadToolBar which loads the toolbar that you designed in the toolbar designer. LoadToolBar function requires a single argument that is the ID for the toolbar. There are various toolbar control styles. Table 3.1 shows different toolbar control styles.

Table 3.1 *Various Toolbar Control Styles*

Style	Style Description
TBSTYLE_LIST	Button text appears to the right of the bitmap image.
TBSTYLE_CUSTOMERASE	Generates a NM_CUSTOMDRAW message when erasing the toolbar and button background, allowing the programmer to choose when and whether to control the background erasing process.
TBSTYLE_FLAT	Creates a flat toolbar. Button text appears under the bitmap image.
TBSTYLE_ALTDRAG	Allows the user to move the toolbar by dragging it while holding down the Alt key.
TBSTYLE_TRANSPARENT	Creates a transparent toolbar.
TBSTYLE_WRAPABLE	Creates a toolbar that can have multiple rows of buttons.
TBSTYLE_TOOLTIPS	Creates a tooltip control that can be used to display descriptive text for the buttons.

For setting the Toolbar Button styles, first you have to create the toolbar and then write the following code:

```
// Find the Black button on the toolbar
iTBCtlID = m_wndColorBar.CommandToIndex(ID_COLOR_BLACK);
if (iTBCtlID >= 0)
{
// Loop through the buttons, setting them to act as radio
buttons
for (i= iTBCtlID; i < (iTBCtlID + 8); i++)
```

```
m_wndColorBar.SetButtonStyle(i,
TBBS_CHECKGROUP);
}
```

NOTES

In this code, the first line uses the CommandToIndex toolbar function to locate the control number of the ID_COLOR_BLACK button. If you design your toolbar based on the order of colors which could be used on the menu, then this may be the first control having an index of 0. Table 3.2 shows the various toolbar button styles.

Table 3.2 *Toolbar Button Styles*

Style	Style Description
TBSTYLE_CHECKGROUP	Creates a button that acts like a radio button, remaining in the pressed state until another button in the group is pressed. This is actually the combination of the TBSTYLE_CHECK and TBSTYLE_GROUP
TBSTYLE_BUTTON	Creates a standard push button.
TBSTYLE_CHECK	Creates a button that acts like a check box, toggling between the pressed and unpressed state.
TBSTYLE_AUTOSIZE	The button's width will be calculated based on the text on the button.
TBSTYLE_GROUP	Create a button that remains pressed until another button in the group is pressed.
TBSTYLE_DROPDOWN	Creates a drop-down list button.

Docking the Toolbar

Following code is used to add the OnCreate function in the CMainFrame class.

```
// Enable docking for the Color Toolbar
m_wndColorBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY); // (AppWizard generated
line)
// Dock the Color Toolbar
DockControlBar(&m_wndColorBar);
```

In the first line of code, there is EnableDocking toolbar function that lets you to enable the toolbar for docking with the frame window. The value passed to this toolbar function must match the value passed in the following EnableDocking function that is called for the frame window. Table 3.3 shows the various Toolbar Docking Styles with style descriptions.

NOTES

Table 3.3 *Toolbar Docking Styles*

Style	Style Description
CBRS_ALIGN_BOTTOM	Allows the toolbar to be docked to the bottom of the view area of the frame window.
CBRS_ALIGN_TOP	Allows the toolbar to be docked to the top of the view area of the frame window.
CBRS_ALIGN_RIGHT	Allows the toolbar to be docked to the right side of the view area of the frame window.
CBRS_ALIGN_LEFT	Allows the toolbar to be docked to the left side of the view area of the frame window.
CBRS_ALIGN_MULTI	Allows multiple toolbars to be floated in a single miniframe window. The toolbar will not be able to dock with the frame.
CBRS_ALIGN_ANY	Allows the toolbar to be docked to any side of the view area of the frame window.

3.3 COMMON CONTROLS

Command Button Control: The Command Button adds a clickable button to the form that may be used to run a specified Check Code block. Here are a few examples of how a Command Button and the Check Code behind the button might be used: Field values are compared, and automatic calculations are performed.

3.3.1 Check Box Control

A checkbox is a windows control that allows the user to change or set the value of an item as true or false. There are various methods in Checkbox class. There are various messages mapping for checkbox control. Following are the steps to draw Check Box.

Step 1: Delete the TODO line and drag one checkbox and one Edit control as shown below.

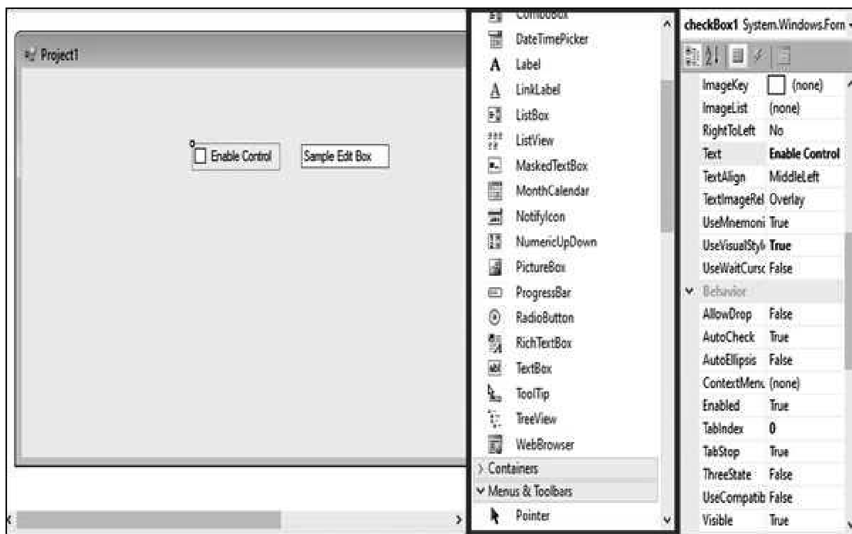


Fig. 3.2 *Creating Check Box Control*

NOTES

Step 2: Right-click on the checkbox and select add variable.

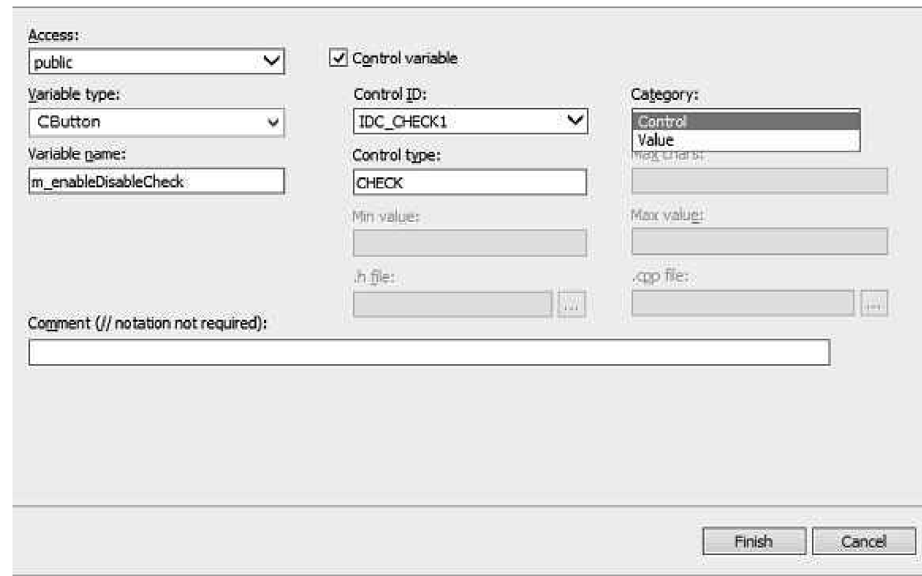


Fig. 3.3 CheckBox-Adding Variables

Step 3: There are different options that can be seen in this dialog box. By default, the CButton variable type is selected for checkbox.

Step 4: similarly, the control ID is also selected by default. Now, select Control in the Category combo box, and in the Variable Name edit box, type m_enableDisableCheck and click Finish.

Step 5: Add Control Variable of Edit control with the settings as shown below.

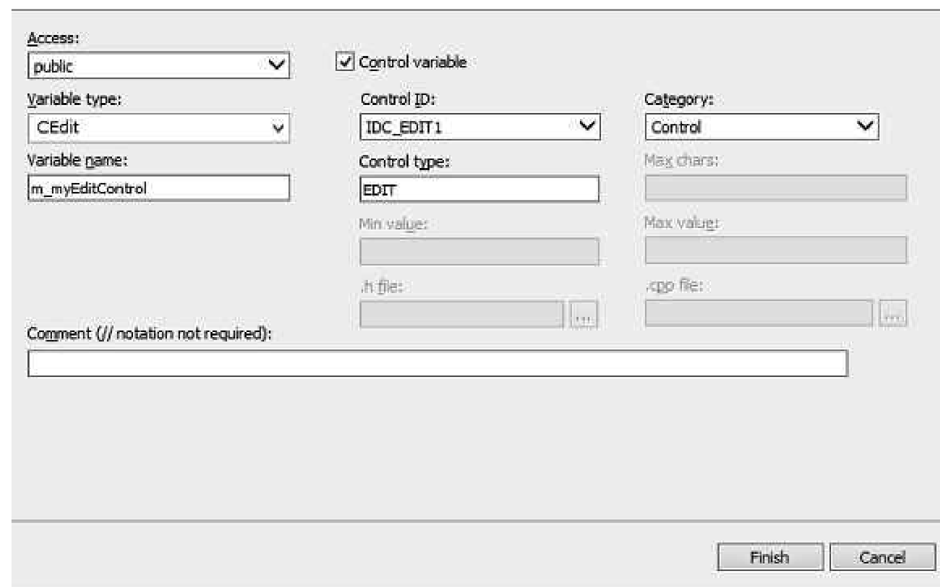


Fig. 3.4 Adding Control Variable of Edit control

Step 6: Click Finish to continue. The code for CheckBox is given below:

```
#include "stdafx.h"
#include "MFCControlManagement.h"
```

NOTES

```
#include "MFCControlManagementDlg.h"
#include "afxdialogex.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialogEx {
   public:
      CAboutDlg();

   // Dialog Data
   #ifdef AFX_DESIGN_TIME
      enum { IDD = IDD_ABOUTBOX };
   #endif
   protected:


virtual void DoDataExchange(CDataExchange* pDX);

 // Implementation
   protected:
      DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialogEx(IDD_ABOUTBOX) {
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX) {
   CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()
CMFCControlManagementDlg::CMFCControlManagementDlg(CWnd*
pParent /* = NULL*/)
   : CDialogEx(IDD_MFCCONTROLMANAGEMENT_DIALOG, pParent),
   m_enableDisableVal(FALSE), m_editControlVal(_T("")) {

   m_hIcon = AfxGetApp()'!LoadIcon(IDR_MAINFRAME);
}
voidCMFCControlManagementDlg::DoDataExchange(CDataExchange*
pDX) {
   CDialogEx::DoDataExchange(pDX);
   DDX_Control(pDX, IDC_CHECK1, m_enableDisableCheck);
```

NOTES

```
    DDX_Control(pDX, IDC_EDIT1, m_myEditControl);
    DDX_Check(pDX, IDC_CHECK1, m_enableDisableVal);
    DDX_Text(pDX, IDC_EDIT1, m_editControlVal);
}

BEGIN_MESSAGE_MAP(CMFCControlManagementDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
                    ON_BN_CLICKED(IDC_CHECK1,
&CMFCControlManagementDlg::OnBnClickedCheck1)
END_MESSAGE_MAP()
// CMFCControlManagementDlg message handlers

BOOL CMFCControlManagementDlg::OnInitDialog() {
    CDialogEx::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL) {
        BOOL bNameValid;
        CString strAboutMenu;
bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
        ASSERT(bNameValid);
        if (!strAboutMenu.IsEmpty()) {
pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // Set the icon for this dialog. The framework does
this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);      // Set big icon
    SetIcon(m_hIcon, FALSE);     // Set small icon

    // TODO: Add extra initialization here
    UpdateData(TRUE);
```

NOTES

```
  if (m_enableDisableVal)
    m_myEditControl.EnableWindow(TRUE);
  else
    m_myEditControl.EnableWindow(FALSE);
  return TRUE; // return TRUE unless you set the focus
to a control
}
void CMFCControlManagementDlg::OnSysCommand(UINT nID,
LPARAM lParam) {
  if ((nID & 0xFFF0) == IDM_ABOUTBOX) {
    CAboutDlg dlgAbout;
    dlgAbout.DoModal();
  }else {
CDialogEx::OnSysCommand(nID, lParam);
  }
}

// If you add a minimize button to your dialog, you will
need the code below
// to draw the icon. For MFC applications using the
document/view model,
// this is automatically done for you by the framework.

void CMFCControlManagementDlg::OnPaint() {
  if (IsIconic()) {
    CPaintDC dc(this); // device context for painting

    SendMessage(WM_ICONERASEBKGND,
      reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
// Center icon in client rectangle
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;
// Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
  }else{
    CDialogEx::OnPaint();
  }

}
// The system calls this function to obtain the cursor to
display while the user drags
```

```
// the minimized window.
HCURSOR CMFCControlManagementDlg::OnQueryDragIcon() {
  return static_cast<HCURSOR>(m_hIcon);
}
void CMFCControlManagementDlg::OnBnClickedCheck1(){
// TODO: Add your control notification handler code here
  UpdateData(TRUE);
  if (m_enableDisableVal)
    m_myEditControl.EnableWindow(TRUE);
   else
    m_myEditControl.EnableWindow(FALSE);
}
```

Step 7: After implementing the above code, the output will be as follows.

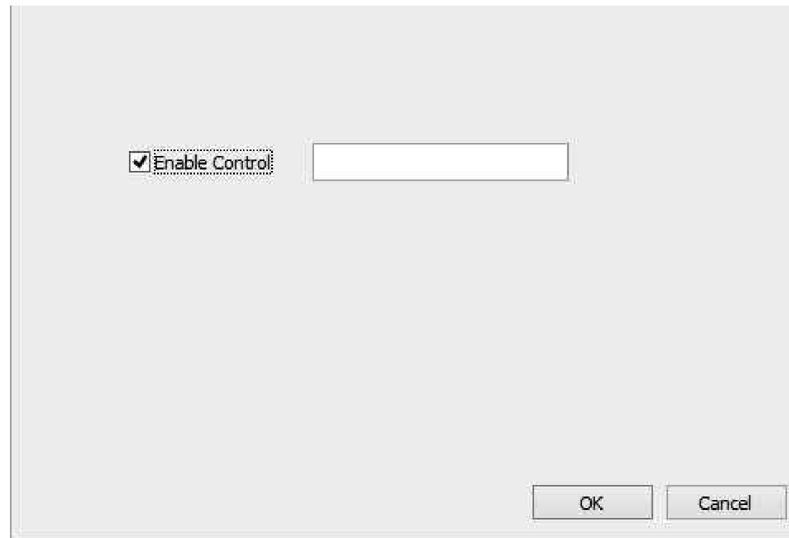


Fig. 3.5 *Check Box Control*

3.3.2 Radio Button Control

A radio button is a control that appears as a dot surrounded by a round box. A radio button is convoyed by one or more other radio buttons that behaves like a group. Table 3.4 shows the list of methods in Radio Button class.

Table 3.4 *Radio Button Classes*

Radio Button Style	Style Description
BN_DISABLE	The framework calls this member function when button is disabled.
BN_DOUBLECLICKED	The framework calls this member function when button is double clicked.
BN_CLICKED	The framework calls this member function when button is clicked.
BN_PAINT	The framework calls this member function when an application makes a request to repaint a button.

Following are the steps for creating Radio Button Control.

Step 1: First, drag a group box and three radio buttons. You should remove the caption of static text control.

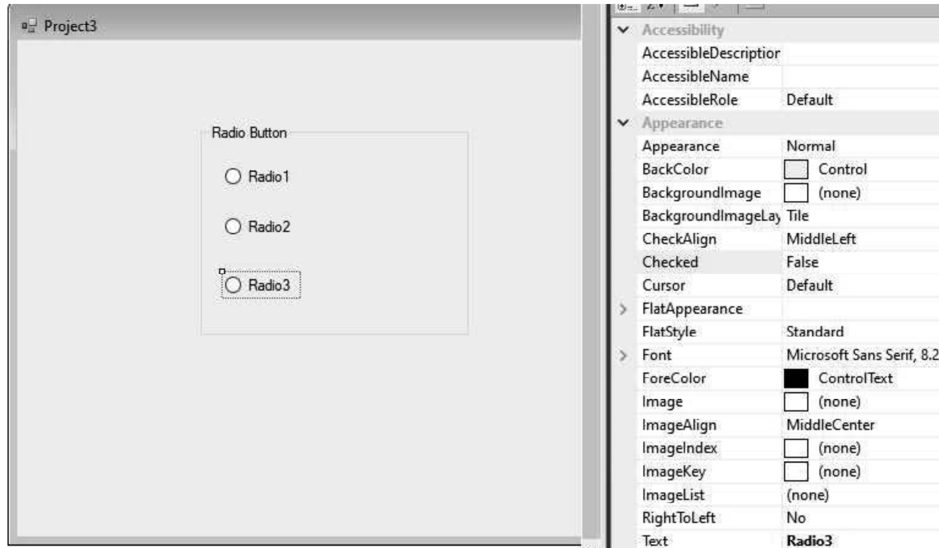


***Fig. 3.6** Radio Button Control*

Step 2: Add the event handler for all the three radio buttons and add the value variable for the static text control.

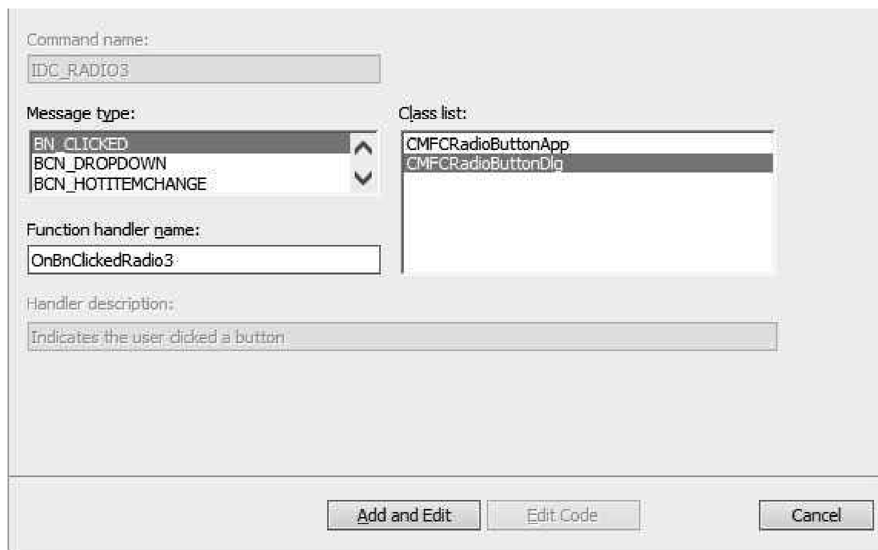


***Fig. 3.7** Radio Button- Adding Event Handler*

Step 3: The code for the implementation of three event handlers for Radio Button is given below:

```
void CMFCRadioButtonDlg::OnBnClickedRadio1() {
   // TODO: Add your control notification handler code
here
  m_strTextControl = _T("Radio Button 1 Clicked");
```

NOTES

```
   UpdateData(FALSE);
}

void CMFCRadioButtonDlg::OnBnClickedRadio2() {
    // TODO: Add your control notification handler code
here
   m_strTextControl = _T("Radio Button 2 Clicked");
   UpdateData(FALSE);
}

void CMFCRadioButtonDlg::OnBnClickedRadio3() {
    // TODO: Add your control notification handler code
here
   m_strTextControl = _T("Radio Button 3 Clicked");
   UpdateData(FALSE);
}
```

Step 4: After compilation and execution of the above code, the output will be as shown below:

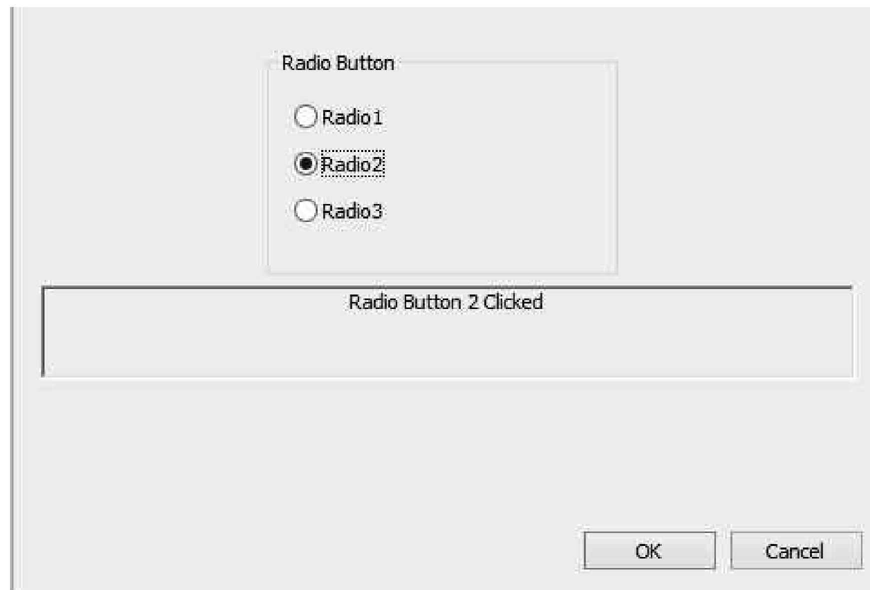


Fig. 3.8 *Radio Button Control with Second Button Selected*

3.3.3 List Box Control

A list box displays a list of items, like filenames, that the user can view and select. A List box is shown with the help of CListBox class. The list of methods in CListBox class as shown in below table 3.5.

Table 3.5 List Box Control Classes

List Box Style	Style Description
LBN_SETFOCUS	The framework calls this member function after gaining the input focus.
LBN_SELCHANGE	The framework calls this member function when selection is changed.
LBN_KILLFOCUS	The framework calls this member function immediately before losing the input focus.
LBN_DBLCLK	The framework calls this member function when list item is double clicked.

Following are the steps to draw List Box:

Step 1: After creation of the project, you will see the TODO line which is the caption of Text Control. Just remove the caption and set its ID to IDC_STATIC_TXT.

Step 2: Drag List Box from the Toolbox as shown in Figure 3.9.

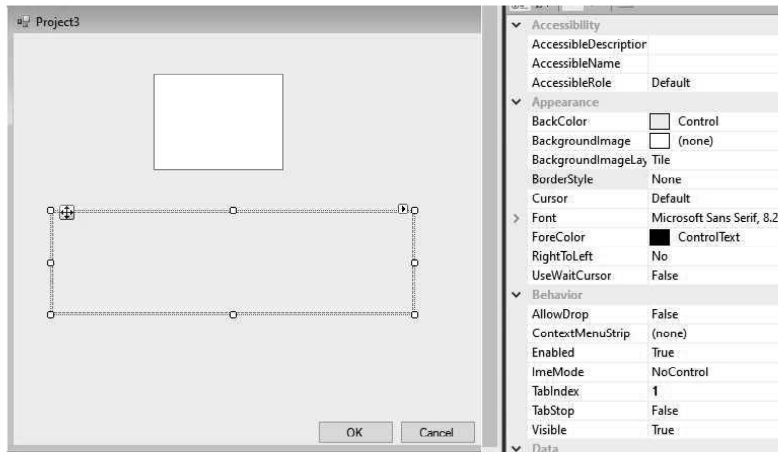


Fig. 3.9 *List Box Control*

Step 3: Add the control and value variable for the Text control as shown in Figure 3.10.

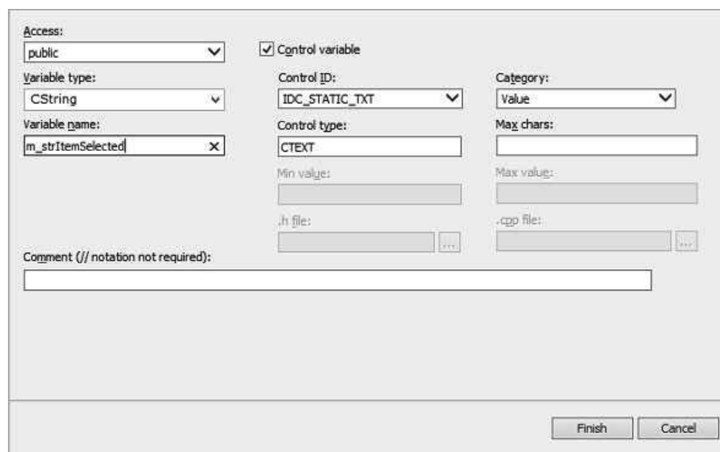


Fig. 3.10 *Add the Control and Value Variable for the Text Control*

Step 4: Add the code given below for the list box:

```
void CMFCListBoxDlg::LoadListBox() {
   CString str = _T("");
   for (int i = 0; i<10; i++) {
      str.Format(_T("Item %d"), i);
      m_listBox.AddString(str);
   }
}

BOOL CMFCListBoxDlg::OnInitDialog() {
   CDialogEx::OnInitDialog();
   // Set the icon for this dialog. The framework does
this automatically
   // when the application's main window is not a dialog
   SetIcon(m_hIcon, TRUE);       // Set big icon
   SetIcon(m_hIcon, FALSE);      // Set small icon
   // TODO: Add extra initialization here
   LoadListBox();
return TRUE; // return TRUE unless you set the focus to a
control
}
void CMFCListBoxDlg::OnLbnSelchangeList1() {
   // TODO: Add your control notification handler code
here
   m_listBox.GetText(m_listBox.GetCurSel(),m_strItemSelected);
   UpdateData(FALSE);
}
```

Step 5: After execution of the above code, the output will be as shown in Figure 3.11 that represents the list of items, where item 4 is selected.

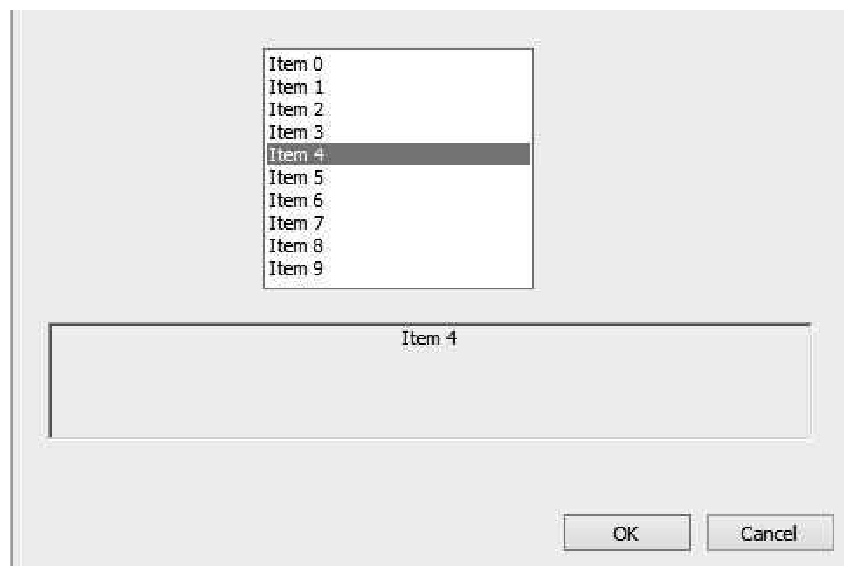


Fig. 3.11 *List Box Control with Item 4 Selected*

3.3.4 Combo Box Control

A combo box consists of a list box combined with either edit control or static control. It enables users to select a predefined value in a list or type their own value in the text box portion of the control. It is extracted with the help of CComboBox class. Table 3.6 shows the list of methods of CComboBox class.

Table 3.6 *Methods of Combo Box Classes*

Combo Box Style	Style Description
CBN_KILLFOCUS	The combo box is losing the input focus.
CBN_DBLCLK	The user double-clicks a string in the list box of a combo box.
CBN_DROPDOWN	The list box of a combo box is about to drop down.
CBN_EDITUPDATE	The edit-control portion of a combo box is about to display altered text.
CBN_EDITCHANGE	The user has taken an action that may have altered the text in the edit control portion of a combo box.
CBN_SELCHANGE	The selection in the list box of a combo box is about to be changed as a result of the user either clicking in the list box or changing the selection by using the arrow keys.
CBN_SETFOCUS	The combo box receives the input focus.

Following are the steps to create the Combo Box Control.

Step 1: Drag a Combo box and remove the caption of static Text control as shown in Figure 3.12.

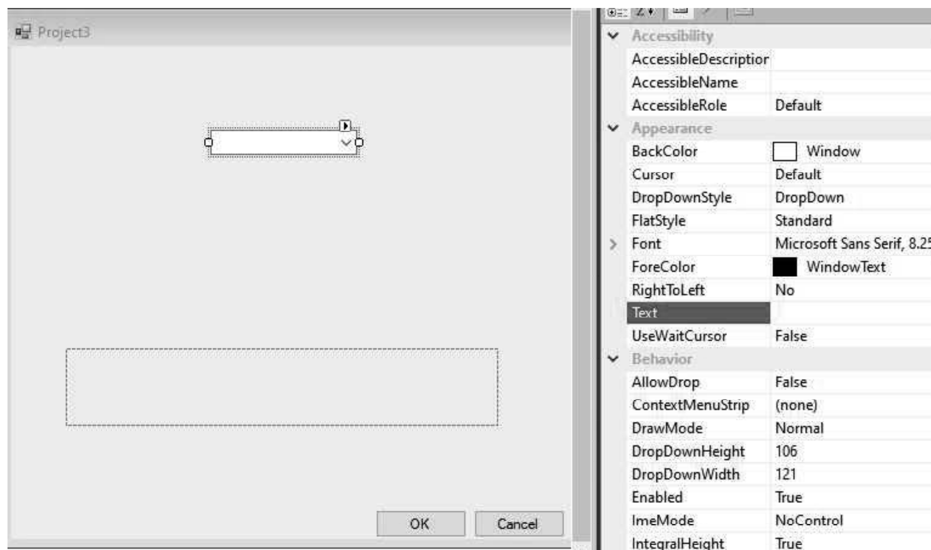


Fig. 3.12 *Combo Box Control*

Step 2: Add a control variable m_comboBoxCtrl for combo box. Also, add the value variable m_strTextCtrl for static Text control.

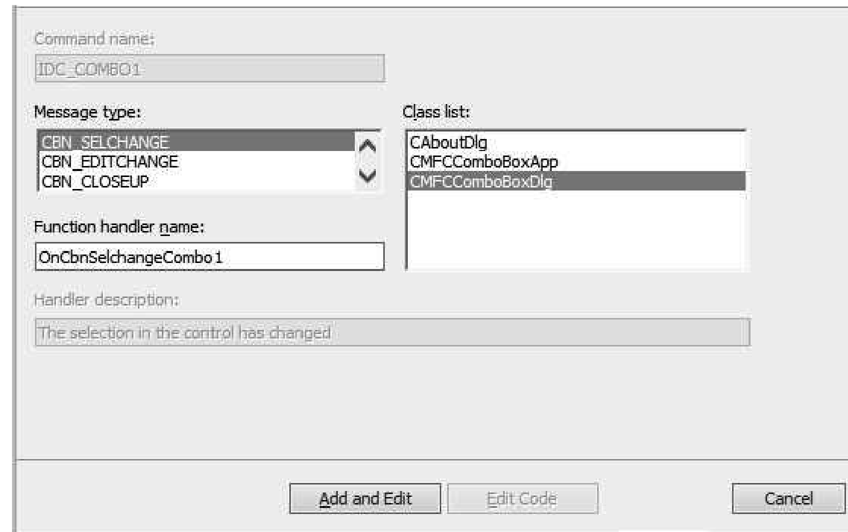


Fig. 3.13 Adding Control and Value variable for Combo Control

Step 3: The code in OnInitDialog() to load the combo box is given below.

```
for (int i = 0; i<10; i++) {
   str.Format(_T("Item %d"), i);
   m_comboBoxCtrl.AddString(str);
}
void CMFCComboBoxDlg::OnCbnSelchangeCombo1() {

   // TODO: Add your control notification handler code
here
   m_comboBoxCtrl.GetLBText(m_comboBoxCtrl.GetCurSel(),
m_strTextCtrl);
UpdateData(FALSE);
}
```

Step 4: After execution of the above code, the output of Combo Box with item 5 selected is as follows.

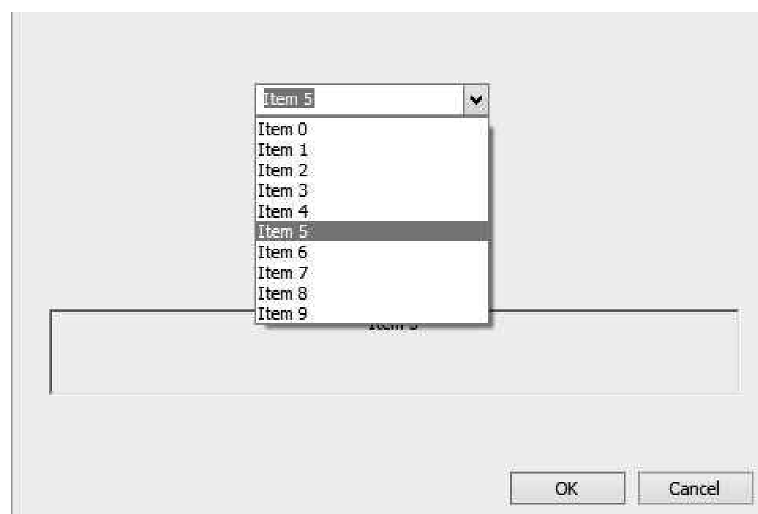


Fig. 3.14 Combo Box with Item 5 Selected

3.3.5 Slider Control

A Slider Control is a window containing tick marks and a slide. You can use either the mouse or the direction keys, when the user moves the slider, the control sends notification messages to indicate the change. There are two types of slider i.e. horizontal and vertical. It is represented in the CSliderCtrl class. Following are the steps to create a slider control.

Step 1: After creating the project, there is TODO line which is the caption of Text Control. You should remove the cCaption and set its ID to IDC_STATIC_TXT.

Step 2: Add a value variable m_strSliderVal for the static Text control.

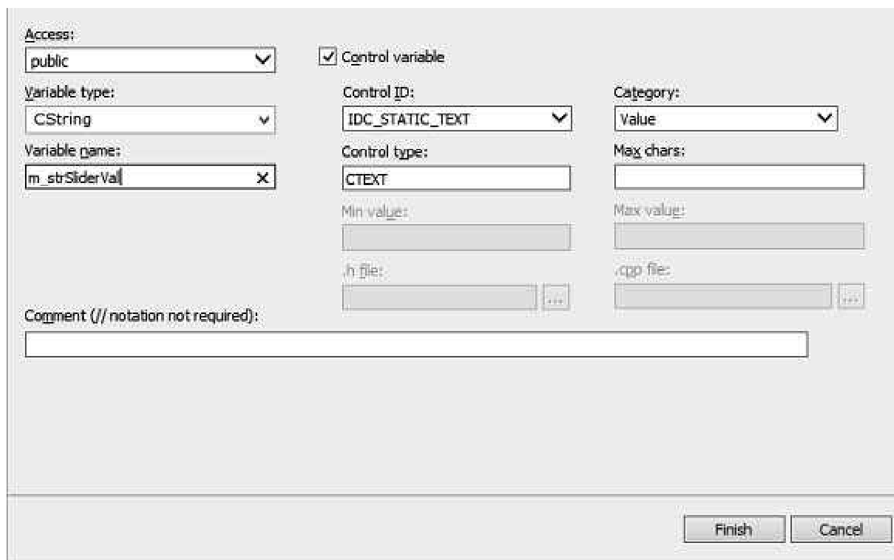


Fig. 3.15 Adding Value Variable for Slider Control

Step 3: Drag the slider control from the Toolbox.

Step 4: Add a control variable m_sliderCtrl for slider as shown in Figure 3.16.

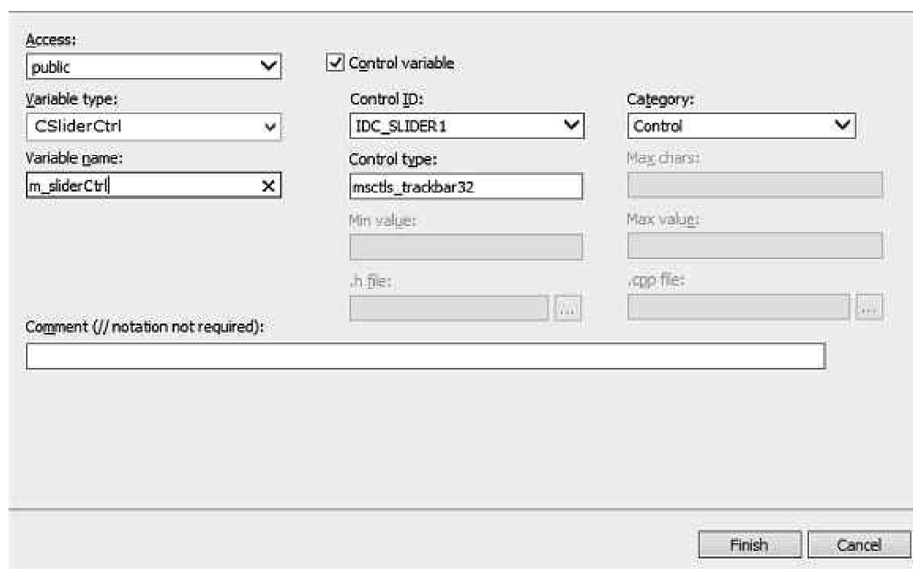


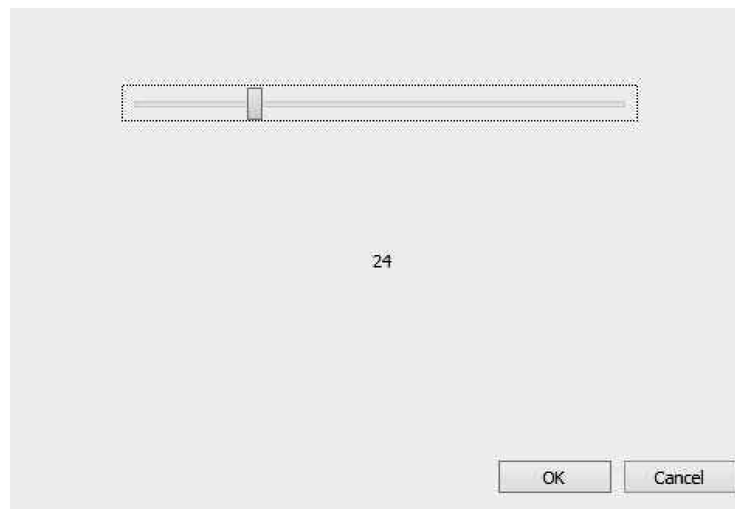
Fig. 3.16 Adding Control Variable for Slider Control

NOTES

```
BOOL CMFCSliderControlDlg::OnInitDialog() {
   CDialogEx::OnInitDialog();
   // Set the icon for this dialog. The framework does
this automatically
   // when the application's main window is not a dialog
   SetIcon(m_hIcon, TRUE);           // Set big icon
   SetIcon(m_hIcon, FALSE);          // Set small icon

   // TODO: Add extra initialization here
   m_sliderCtrl.SetRange(0, 100, TRUE);
   m_sliderCtrl.SetPos(0);
m_strSliderVal.Format(_T("%d"), 0);

   return TRUE; // return TRUE unless you set the focus
to a control
}
void CMFCSliderControlDlg::OnHScroll(UINT nSBCode, UINT
nPos, CScrollBar* pScrollBar) {
   // TODO: Add your message handler code here and/or
call default
if (pScrollBar == (CScrollBar *)&m_sliderCtrl) {
      int value = m_sliderCtrl.GetPos();
     m_strSliderVal.Format(_T("%d"), value);
      UpdateData(FALSE);
   }else {
     CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
   }
}
```

Step 5: Add the following code to draw the slider. You should declare the slider and static text control inside the OnInitDialog() function.

Step 6: When you run and execute the above code, the output will be as follows.



Check Your Progress

1. Define the term GUI.
2. What is command button control?
3. What is the use of check box control?
4. State about the radio button control.
5. What is slider control?

3.4 BUILDING BLOCKS OF HELP

There are some functions which are building blocks of Help. A global function, ConstructElements() is called when you call the SetSize() function member of an array collection. This function is used to allocate memory for the number of elements that you want to store in the array collection. This function is also known as helper function because it helps in setting the size of the array. The default value of this function sets the contents of the assigned memory to zero. This function don't call a constructor for the object class. ConstructElements() is also called by the member function InsertAt(). This member function inserts one or more elements at a specific index position in an array. The CArray collection class members remove the elements, call the helper function DestructElements(). If the object construction allocates any memory on the heap, then override this function to release the memory appropriately.

The parameters of CList collection class templates are similar as those for the CArray template:

```
CList<ObjectType, ObjectType&> aList;
```

When you declare a list collection, there is requirement to supply two arguments to the template i.e. object is to be specified in function arguments and the type of object to be stored. A list collection can be specified to store the points which are used to identifying a curve object as shown below.

```
CList<CPoint, CPoint&> PointList;
```

In the above code, PointList stores the CPoint objects. It is passed as class by reference to functions.

Helper functions are also used by CMap. The map collection classes uses a global function named as HashKey(). It is represented in template given below:

```
template<class ARG_KEY>
UINT HashKey(ARG_KEY key);
```

This template function is used to convert the key value to a hash value of type UINT (equivalent to unsigned int). There could be various techniques used for hashing. These techniques can differ depending on the type of data being used as a Key. The possible number of elements to be stored shows the number of unique hash values. The most common method for hashing a numeric key value is to calculate the hash value as the value of the key modulo N, where N is the number

of different values. For example, consider if you want to store up to 200 different entries in a map using a key value, Key. This can be done with the following statement.

HashValue = Key%201;

This shows the result in values for the HashValue. This value ranges in between 0 and 200 which is exactly as you required to calculate the address for an entry.

3.5 PROPERTY PAGES AND SHEETS

A property sheet is a dialog box that consists of property pages. It is also known as a tab dialog box. It is enclosed on a page with the help of a tab. To select a set of controls, you can click a tab in the property sheet. It is based on a dialog template resource and consists of various controls. Figure 3.17 shown the way a property page is created.

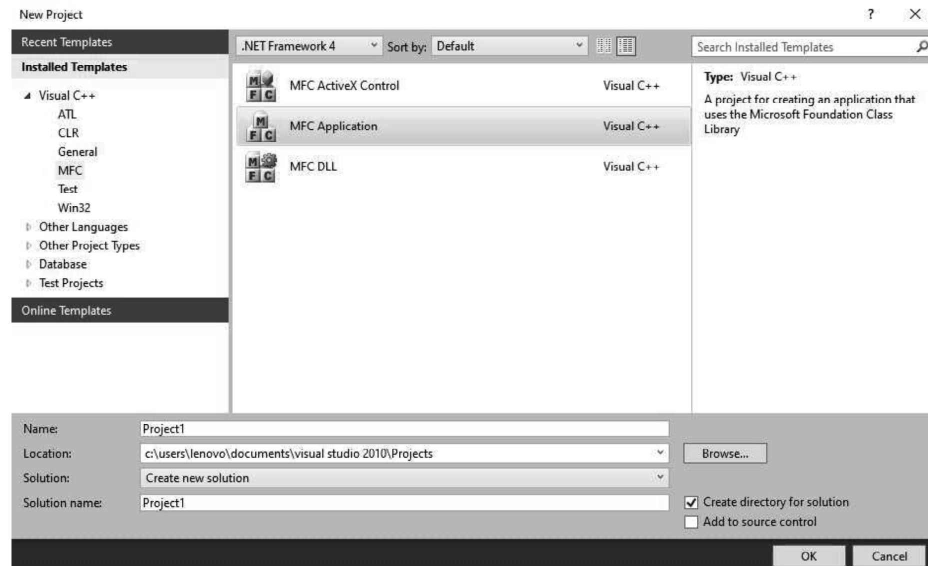


Fig. 3.17 *MFC Property Page*

After creating the project, it is required to add some property pages. By displaying the Add Resource dialog box, it is easy to create resources for property pages. This is done by expanding the Dialog node and selecting one of the IDD_PROPPAGE_X items. Following are the steps for creating the Property pages.

Step 1: Right-click on the project and select Add '! Resources in solution explorer.

Step 2: Select the IDD_PROPPAGE_LARGE and click NEW as shown below.

NOTES

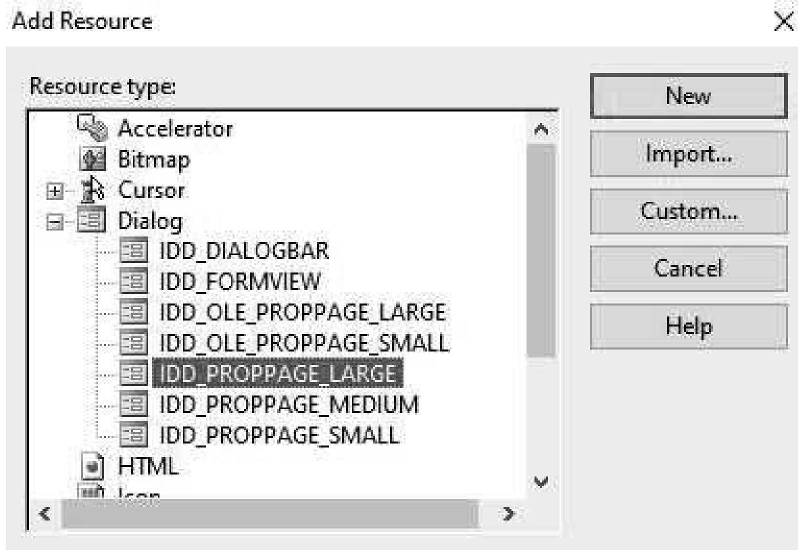


Fig. 3.18 *Add Resources in MFC Property Sheet*

Step 3: Now change ID and caption of this property page to **IDD_PROPPAGE_1** and **Property Page 1** respectively as shown below.

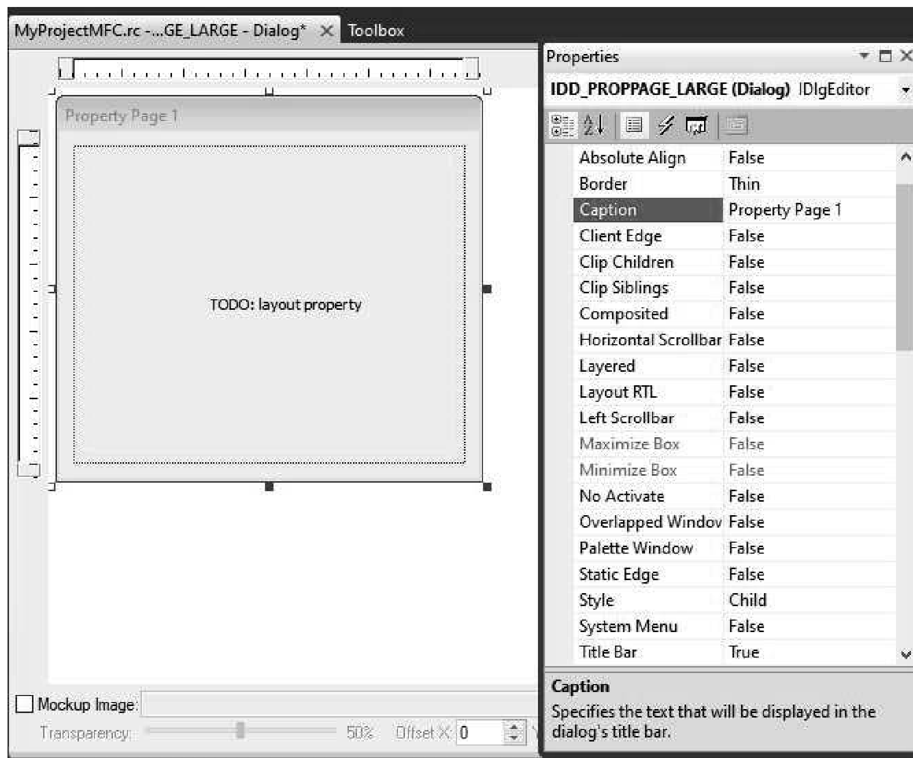


Fig. 3.19 *Property Page with ID and Caption*

NOTES

Step 4: Right-click on the property page in designer window as shown below.

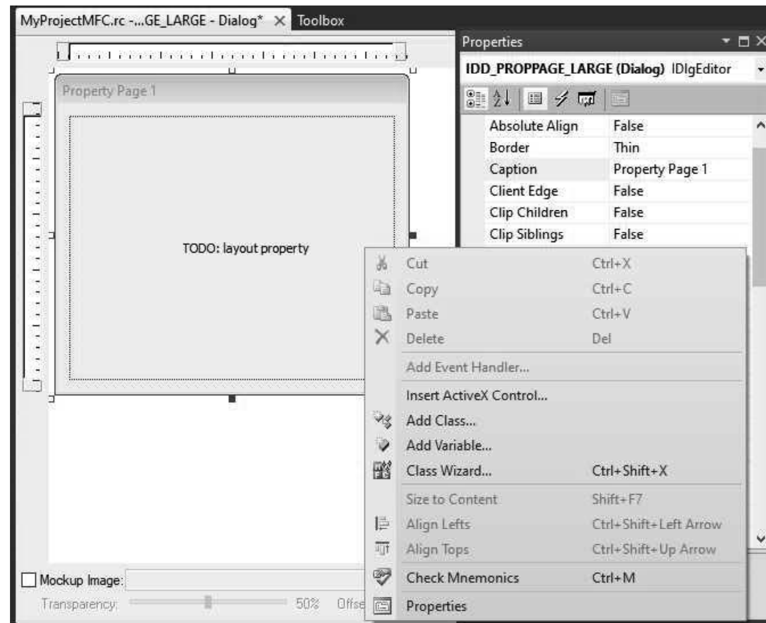


Fig. 3.20 *Property Page with Add Class option*

Step 5: Select the Add Class option as shown below.

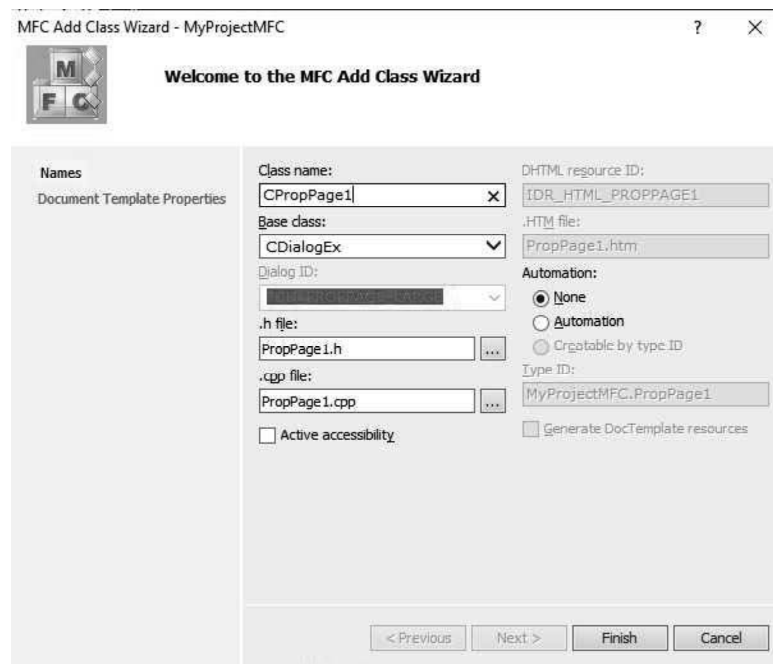


Fig. 3.21 *MFC Add Class Wizard*

Step 6: Enter the class name and select CPropertyPage from base class dropdown list.

Step 7: Click Finish to continue.

Step 8: Add one more property page with ID IDD_PROPPAGE_2 and caption as Property Page 2. Follow the same steps as above.

Step 9: Two property pages have been created. We required a property sheet to implement its functionality.

The Property Sheet groups the property pages together and keeps it as entity. The steps to create a property sheet are as follows:

Step 1: Right-click on your project and select Add'! Class menu options as shown below.

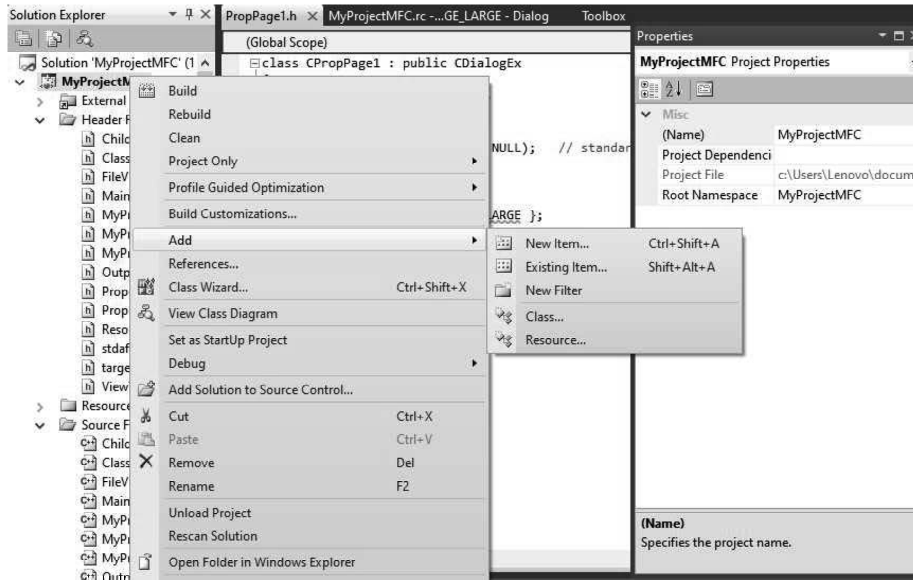


Fig. 3.22 Adding new Class for Creating Property Sheet

Step 2: From the left pane, select Visual C++ '! MFC and MFC Class in the template pane and click Add as shown below.

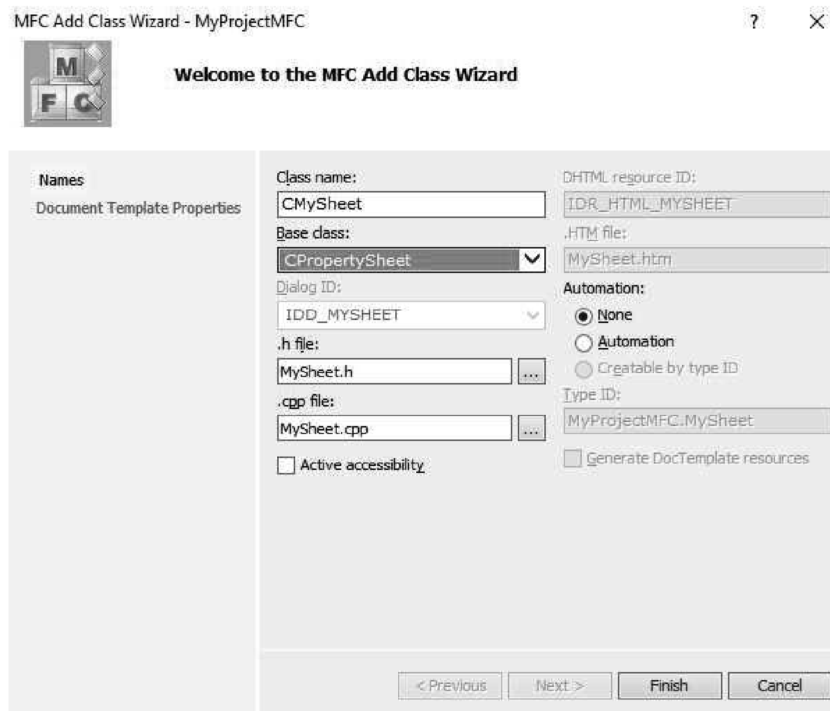


Fig. 3.23 MFC Add Class Wizard

Step 3: Enter the class name and select CPropertySheet from base class dropdown list and click finish to continue.

Step 4: There is a requirement of the following changes in our main project class to launch this property sheet.

Step 5: Add the following code given in CMFCPropSheetDemo.cpp file.

```
#include "MySheet.h"
#include "PropPage1.h"
#include "PropPage2.h"
//Modify the CMFCPropSheetDemoApp::InitInstance() method
CMySheet mySheet(L"Property Sheet Demo");
CPropPage1 page1;
CPropPage2 page2;
mySheet.AddPage(&page1);
mySheet.AddPage(&page2);
m_pMainWnd = &mySheet;
INT_PTR nResponse = mySheet.DoModal();
// The complete implementation of CMFCPropSheetDemo.cpp
file
// MFCPropSheetDemo.cpp : Defines the class behaviors
for the application.
//
#include "stdafx.h"
#include "MFCPropSheetDemo.h"
#include "MFCPropSheetDemoDlg.h"
#include "MySheet.h"
#include "PropPage1.h"
#include "PropPage2.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
// CMFCPropSheetDemoApp
BEGIN_MESSAGE_MAP(CMFCPropSheetDemoApp, CWinApp)
  ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()
// CMFCPropSheetDemoApp construction
CMFCPropSheetDemoApp::CMFCPropSheetDemoApp() {
   // support Restart Manager
m_dwRestartManagerSupportFlags =
AFX_RESTART_MANAGER_SUPPORT_RESTART;
   // TODO: add construction code here,
  // Place all significant initialization in InitInstance
}
// The one and only CMFCPropSheetDemoApp object
CMFCPropSheetDemoApp theApp;
```

NOTES

```
// CMFCPropSheetDemoApp initialization
BOOL CMFCPropSheetDemoApp::InitInstance() {
  // InitCommonControlsEx() is required on Windows XP if
an application
  // manifest specifies use of ComCtl32.dll version 6 or
later to enable
  // visual styles. Otherwise, any window creation will
fail.
  INITCOMMONCONTROLSEX InitCtrls;
  InitCtrls.dwSize = sizeof(InitCtrls);
  // Set this to include all the common control classes
you want to use
   // in your application.
  InitCtrls.dwICC = ICC_WIN95_CLASSES;
  InitCommonControlsEx(&InitCtrls);
CWinApp::InitInstance();
 AfxEnableControlContainer();
  // Create the shell manager, in case the dialog contains
 // any shell tree view or shell list view controls.
CShellManager *pShellManager = new CShellManager;
// Activate "Windows Native" visual manager for enabling
themes in MFC controls
CMFCVisualManager::SetDefaultManager
(RUNTIME_CLASS(CMFCVisualManagerWindows));
  // Standard initialization
   // If you are not using these features and wish to
reduce the size
   // of your final executable, you should remove from
the following
   // the specific initialization routines you do not
need
   // Change the registry key under which our settings
are stored
  // TODO: You should modify this string to be something
appropriate
   // such as the name of your company or organization
     SetRegistryKey(_T("Local  AppWizard-Generated
Applications"));
  CMySheet mySheet(L"Property Sheet Demo");
   CPropPage1 page1;
   CPropPage2 page2;
mySheet.AddPage(&page1);
  mySheet.AddPage(&page2);
   m_pMainWnd = &mySheet;
  INT_PTR nResponse = mySheet.DoModal();
   if (nResponse == IDOK) {
```

NOTES

```
        // TODO: Place code here to handle when the dialog
is
        // dismissed with OK
    }else if (nResponse == IDCANCEL) {
        // TODO: Place code here to handle when the dialog
is
        // dismissed with Cancel
    }else if (nResponse == -1) {
        TRACE(traceAppMsg, 0, "Warning: dialog creation
failed,
        so application is terminating unexpectedly.\n");
        TRACE(traceAppMsg, 0, "Warning: if you are using
MFC controls on the dialog,
                            you    cannot    #define
_AFX_NO_MFC_CONTROLS_IN_DIALOGS.\n");
    }

    // Delete the shell manager created above.
    if (pShellManager != NULL) {
        delete pShellManager;
    }

    // Since the dialog has been closed, return FALSE so
that we exit the
    // application, rather than start the application's
message pump.
return FALSE;
}
```

Step 6: After compilation and execution of the above code, the output will be as shown below. It shows the dialog box containing two property pages.

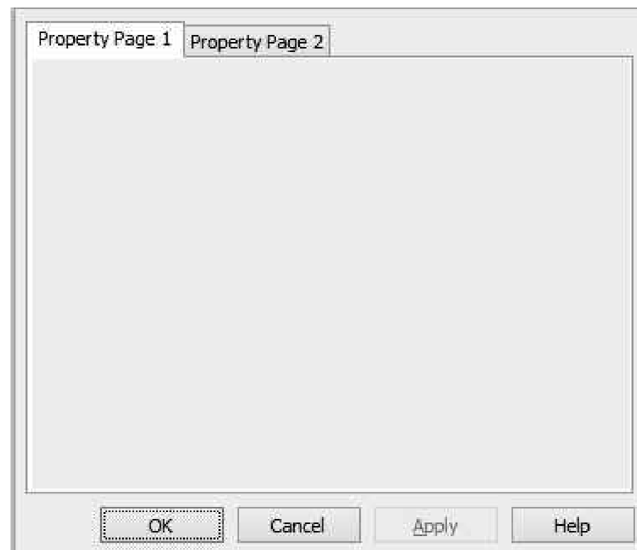


Fig. 3.24 *MFC Property Sheet Output*

NOTES

Check Your Progress

6. How will you define the building blocks of help?
7. What is a property sheet?

3.6 ANSWER TO 'CHECK YOUR PROGRESS'

1. Windows makes the computers easier to use and learn and that was the driving objective behind the development of Graphical User Interfaces (GUI). The GUI designers stated that a standard set of menus should be used by all applications and these menus should be organized in a uniform manner.
2. The Command Button adds a clickable button to the form that may be used to run a specified Check Code block. Here are a few examples of how a Command Button and the Check Code behind the button might be used: Field values are compared, and automatic calculations are performed.
3. A checkbox is a windows control that allows the user to change or set the value of an item as true or false.
4. A radio button is a control that appears as a dot surrounded by a round box. A radio button is convoyed by one or more other radio buttons that behaves like a group.
5. A Slider Control is a window containing tick marks and a slide. By using either the mouse or the direction keys, when the user moves the slider, the control sends notification messages to indicate the change.
6. There are some helper functions() which are building blocks of Help. When you call the SetSize() function member of an array collection, a global function, ConstructElements(), is called. This function is used to allocate memory for the number of elements that you want to store in the array collection. This function is also known as helper function because it helps in setting the size of the array collection. The default value of this function sets the contents of the assigned memory to zero.
7. A property sheet is a dialog box that consists of property pages. It is also known as a tab dialog box. It is enclosed on a page with the help of a tab. To select a set of controls, you can click a tab in the property sheet. It is based on a dialog template resource and consists of various controls.

3.7 SUMMARY

- The GUI designers stated that a standard set of menus should be used by all applications and these menus should be organized in a uniform manner.
- The Command Button adds a clickable button to the form that may be used to run a specified Check Code block. Here are a few examples of how a Command Button and the Check Code behind the button might be used: Field values are compared, and automatic calculations are performed.

- A checkbox is a windows control that allows the user to change or set the value of an item as true or false.
- A radio button is a control that appears as a dot surrounded by a round box. A radio button is convoyed by one or more other radio buttons that behaves like a group.
- A list box displays a list of items, like filenames, that the user can view and select. A List box is shown with the help of CListBox class.
- A combo box consists of a list box combined with either a edit control or static control. It is denoted with the help of CComboBox class.
- A Slider Control is a window containing tick marks and a slide. By using either the mouse or the direction keys, when the user moves the slider, the control sends notification messages to indicate the change.
- Helper function helps in setting the size of the array collection. The default value of this function sets the contents of the assigned memory to zero.
- A property sheet is a dialog box that consists of property pages. It is also known as a tab dialog box. It is enclosed on a page with the help of a tab. To select a set of controls, you can click a tab in the property sheet. It is based on a dialog template resource and consists of various controls.

3.8 KEY TERMS

- **Check Box Control:** A checkbox is a windows control that allows the user to change or set the value of an item as true or false.
- **Radio Button Control:** A radio button is a control that appears as a dot surrounded by a round box. A radio button is convoyed by one or more other radio buttons that behaves like a group.
- **Slider Control:** A slider control is a window containing tick marks and a slide. By using either the mouse or the direction keys, when the user moves the slider, the control sends notification messages to indicate the change.
- **Tab Dialog Box:** A property sheet is a dialog box that consists of property pages. It is also known as a tab dialog box. It is enclosed on a page with the help of a tab. To select a set of controls, you can click a tab in the property sheet. It is based on a dialog template resource and consists of various controls.

3.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What do you understand by the status bars and tool bars?
2. What are some of the common controls?
3. How will you define the building blocks of help?
4. What do you mean by the property pages and sheets?

Long-Answer Questions

1. Discuss briefly about the status bars and tool bars with the help of examples.
2. What do you understand by the common controls? Discuss the various methods in checkbox class.
3. Discuss about the building blocks of help using approprite example.
4. Explain briefly about the property pages and sheets. Give appropriate examples.

3.10 FURTHER READING

Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.

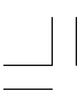
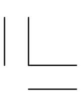
Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.

Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.

Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6*, 1st Edition. New Delhi: BPB Publications.

Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.

Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 4 COMMON CONTROLS

NOTES

Structure

4.0 Introduction
4.1 Objectives
4.2 ActiveX and OLE
4.3 ActiveX and COM
4.3.1 Creating MFC Project to Develop Car Component
4.4 ActiveX Control Macros
4.5 Building an ActiveX Server Application
4.6 Building ActiveX Control
4.6.1 Creating ActiveX Control Container Application
4.6.2 ActiveX Control Methods
4.6.3 ActiveX Events
4.7 Answers to 'Check Your Progress'
4.8 Summary
4.9 Key Terms
4.10 Self-Assessment Questions and Exercises
4.11 Further Reading

4.0 INTRODUCTION

ActiveX controls is an Object Linking and Embedding (OLE) like compound graphical user interface component that transfers the data between the server and the container applications. The earlier COleDoc object is now called the ActiveX Document object. This unit introduces ActiveX control, various ActiveX technologies, creating an ActiveX server program, adding properties, events and event handlers in the ActiveX control.This unit also explains how to access an ActiveX control in the container application.

Component Object Model (COM) is a binary interface standard for software componentry. It was introduced by Microsoft in 1993, and is used to enable interprocess communication and dynamic object creation. The term COM is an umbrella term that covers the OLE, OLE Automation, ActiveX, COM+ and DCOM technologies. COM is a software architecture that allows the components made by different software vendors to be combined into a variety of applications. It lays the standard for component interoperability, is not dependent on a specific programming language, is available on multiple platforms and is extensible.

In this unit, you will learn about the ActiveX and OLE, ActiveX, COM (Component Object Model), ActiveX control macros, building an ActiveX server application, building ActiveX control, ActiveX control methods and ActiveX events.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basic of ActiveX and OLE
- Explain about the ActiveX

- Learn about the COM (Component Object Model)
- Analyze the ActiveX control macros
- Elaborate on the building an ActiveX server application
- Discuss about the building ActiveX control
- Define the ActiveX control methods
- Illustrate the ActiveX events

4.2 ACTIVEX AND OLE

ActiveX Control

An ActiveX control is a refillable software component based on the 'Component Object Model (COM)' which provides support to an extensive diversity of OLE functionality. ActiveX can certainly be modified to use in various other software as required.

ActiveX controls are developed for 'ActiveX control containers'. It is also used in applications, which is hosted in the Internet, i.e. in the World Wide Web (WWW) pages. ActiveX controls can either be created with Microsoft Foundation Class Library (MFC) or with the Active Template Library (ATL).

An ActiveX control has the ability for drawing itself and its particular window. It can also react to various triggered events (e.g. clicking of a mouse button). Additionally, we can manage ActiveX control via an interface which consists of various properties and methods same as in automation objects.

ActiveX controls are developed to cater a variety of activities like to access various databases, data monitoring, graphing, etc. ActiveX controls are ported seamlessly in various application during development. Apart from the ability of portability, ActiveX controls provisions characteristics which does not presented earlier in ActiveX controls. For example, establishing compatibility with prevailing OLE containers, capability for integrating their associated menus with the menus existing in the OLE container, etc. Apart from this, an ActiveX control completely provide support to automation of various activities. This permits the control to use its read\write attributes and set of methods which is normally called by the control user.

You can create windowless ActiveX controls and controls that only create a window when they become active. Windowless controls increase the pace of the display, the associated application and enables the possibility of transparent and nonrectangular controls. ActiveX control properties can also be loaded in an asynchronous manner.

An ActiveX control is deployed as a 'process server' and can also be used in OLE containers. The complete set of features is an ActiveX control is available when it is used within an OLE container designed to be aware of ActiveX controls. This type of container is known as "control container". The function of an ActiveX control via the control properties and methods receives notifications from the ActiveX control in the form of events. This instance is illustrated in the following figure.

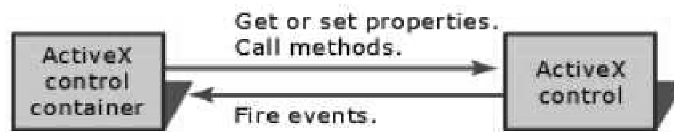


Fig. 4.1 Interaction between an ActiveX Control Container and a Windowed ActiveX Control

NOTES

Various ActiveX Components

- **ActiveX Component:** This ActiveX object is used by supplementary programs.
- **ActiveX Control:** This is a distinct type of ActiveX component which acts as a control window.
- **ActiveX Container:** This is a program which accepts ActiveX controls or document related objects.
- **ActiveX Server:** It is a DLL or executable program which caters the facilities of one or additional ActiveX components.
- **ActiveX Client:** This is a program which use ActiveX component.
- **ActiveX Document Object:** This is a document which is used to establish links or can be embedded within an ActiveX container.

The viewpoint of an ActiveX client, an ActiveX component is fundamentally and table of function pointers is known as a virtual table or vtable. A vtable comprises of an interface for an ActiveX component and an associated component which can consists of various interfaces (vtables) for multiple objectives. When an ActiveX client requires a service from corresponding ActiveX component, it normally asks component member functions using the pointers which are given in the interface's vtable as shown in Figure 4.2.

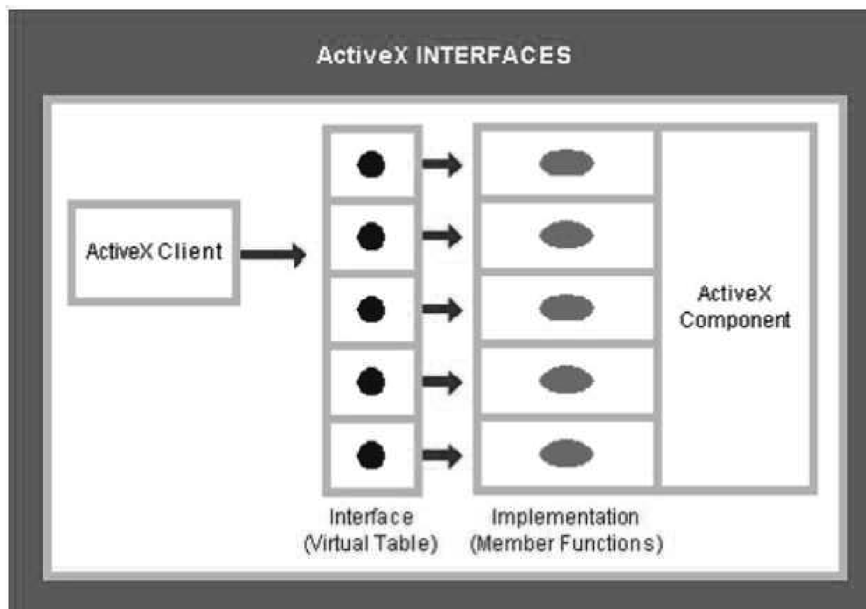


Fig. 4.2 ActiveX Component Interfaces

NOTES

Internet-Centric Architecture

For Internet oriented application, ActiveX components are precisely embedded within an associated HTML container. HTML (Hyper Text Markup Language) is the default standard language for applications hosted on the internet. In other words, HTML is used to create Web pages. It is also used to establish interfaces and perform the required accumulation of respective applications from various components. A generic ActiveX Internet-centric application architecture is shown in Figure 4.3.

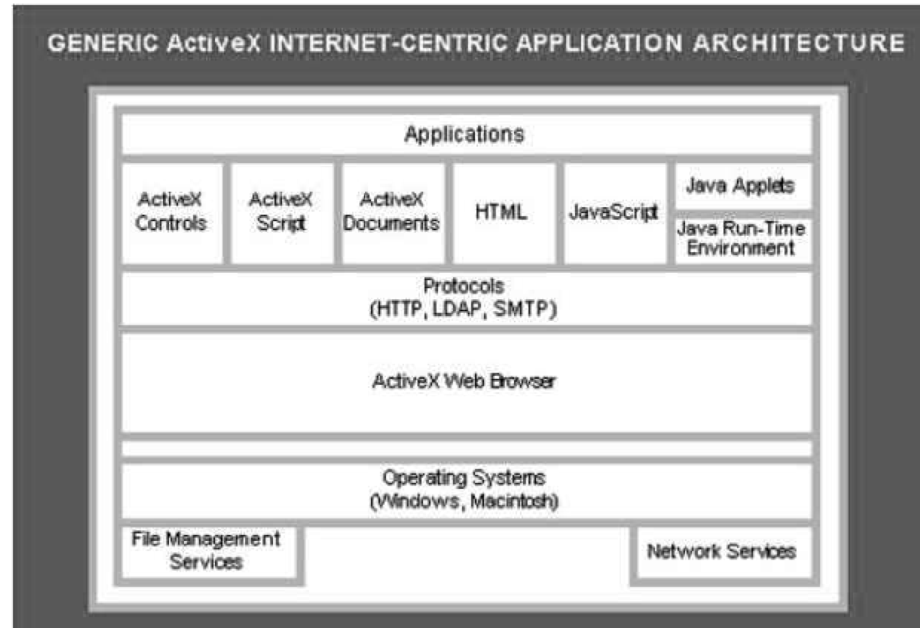


Fig. 4.3 *Generic ActiveX Internet Centric Application Architecture*

The browser framework comprises of the following layers:

- **HTML Application**: It contains HTML explanations, embedded ActiveX controls, ActiveX scripts, ActiveX documents, Java applets and JavaScript scripts.
- **Java Run-Time Environment**: It is the virtual machine (also called Java Virtual Machine – JVM) which translates the byte codes of Java applet.
- **Protocols:** It is a set of rules which are used for communication and collaboration purposes. Examples are - HTTP for document and application retrieval; SMTP, IMAP4 and POP3 for sending and receiving messages; and NNTP for computing purpose within the workgroup.

ActiveX is not at all a programming language. It is a protocol or a set of rules which defines how an application should share data. ActiveX control is a dynamic link library. It function as a COM server and is used to embed the host application in the container.

The code of ActiveX controls is reusable and can be reused in another programming language without any alteration and the net result is the same as using Windows common controls. For example: ActiveX controls which are programmed in VC++ is applicable to VB without any further alteration.

NOTES

ActiveX Filtering

Browsers can be used to deactivate the ActiveX controls for various reasons. If ActiveX are filtered, browsers will restrict installing apps which uses ActiveX. This is certainly a perfect process for safe browsing. For example, in case the ActiveX Filtering is enabled, videos, games, and other applications will never work.

The following steps should be followed for managing ActiveX settings in Internet Explorer.

1. In Internet Explorer, go to ”Tools” button and select ”Internet options”.
2. On the ”Security” tab, select ”Custom level”, and then below ”ActiveX controls and plug-ins”, perform one the following:
   - o Allow ”Automatic prompting for ActiveX” by enabling it.
   - o Allow Internet Explorer to ”Display video and animation on a webpage that doesn’t use external media player” by enabling it.
   - o Allow Internet Explorer to ”Download signed ActiveX controls” by enabling it. “Prompt” can also be selected if it is required to be prompted every time a notification is required before downloading.
   - o Allow Internet Explorer to ”Run ActiveX controls and plug-ins” by enabling it. “Prompt” can also be selected if it is required to be prompted every time a notification is required before downloading.
   - o Allow Internet Explorer to ”Script ActiveX controls marked safe for scripting” by enabling it. “Prompt” can also be selected if it is required to be prompted every time a notification is required before downloading.

Figure 4.4 shows security settings. All other browser containing similar settings can be managed accordingly.

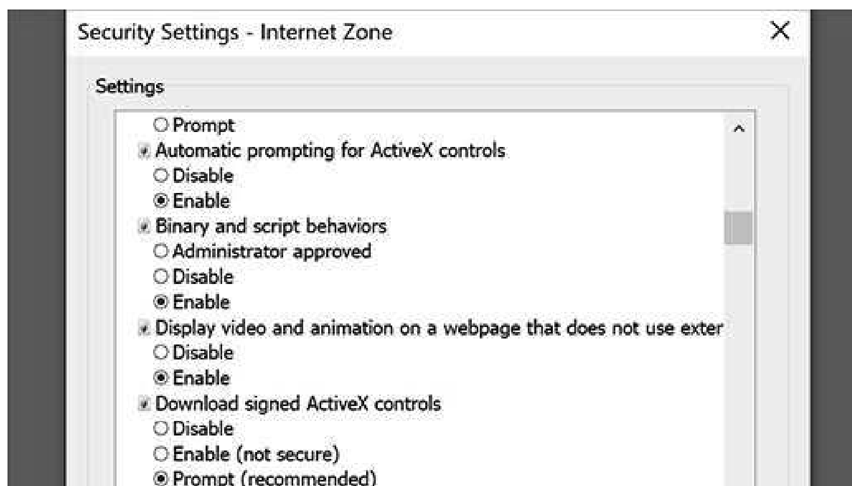


Fig. 4.4 *Setting up the ActiveX control in browser*

3. Select ”OK” and again select ”OK”

Object Linking and Embedding (OLE)

Object Linking and Embedding (OLE) is a technology from Microsoft. It is used to share various application data and objects, formed in various formats from

numerous sources. The term “Linking” creates a connectivity between various objects. The term “Embedding” performs the insertion or appending of data within an application.

NOTES

OLE is used in the area of complex document management system. It is also used for transferring data between applications either via drag-and-drop process or using clipboard related operations. An OLE object can be displayed an icon on the desktop. By clicking twice on the OLE icon, the icon either exposes the corresponding object application or requests the user identify and appropriate application using which the OLE object can be edited.

Applications supporting OLE are as follows:

- Microsoft Word, Excel and PowerPoint
- Corel WordPerfect
- Adobe Acrobat
- AutoCAD
- Multimedia applications such as photos, audio/video clips and PowerPoint presentations.

The main advantages of OLE are as follows:

- In case, changes are performed in source data, the same is available to the clients i.e., the updated data is always presented to the client.
- The primary application is not required to be in place for editing the associated data in the object. External editors can also be used.
- All applications use the same interface to edit OLE data.
- The user can choose his / her favourite editor and the same can be used to modify the object data
- OLE is all about linking and embedding. Hence, no separate data is stored with the clients, disk space is saved.

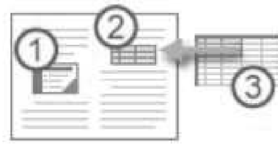
Basic Differences between Linked Objects and Embedded Objects

We need to understand the following to understand the difference between linked objects and embedded objects. They are:

- Where is the data stored?
- How the updation of data is accomplished after placing in the target document?

For example, a MS Excel graph is inserted into a MS PowerPoint presentation. This is accomplished either via a link to the object or pasting a copy of the object in the presentation. Objects can be inserted directly for those application which supports OLE technology.

Let us consider the above example, where we are preparing a project status using MS PowerPoint and it contains data which is existing in a separate spreadsheet. If linking the presentation is performed to the spreadsheet, the data in the presentation gets updated automatically when the spreadsheet gets updated. In case the spreadsheet is embedded within the presentation, the presentation contains just a static replica of the spreadsheet data. Let us see the Figure 4.5.



NOTES

Fig. 4.5 Spreadsheet Data

Where,

1. Object which is embedded
2. Object which is linked
3. Data file which is the source of all data

Linked objects

In case of linked object, data gets updated automatically once the source file is updated. The source file contains the data which is linked. The presentation file (with reference to our example), only preserves the path or location of the source spreadsheet. The link displays the data as per the defined path of the data file. In ideal case, linked objects are used for the data set which are small in size. Linking is certainly beneficial in case it is required to consider data which is stored as standalone.

Embedded objects

When an Excel object is embedded, data in the presentation will never change in case the source data is modifies. Once embedded the objects itself gets converted to be a part of the presentation file itself and this is the reason, when the source file is modified, it is not reflected into the presentation.

Use Case

A multi stored building was built at Mumbai. This was for an organization whose head office was at Australia. The design of the building and its interiors was drafted in AutoCAD file. During the construction of this hotel, the engineers and higher officials from Australia used to conduct a meeting using video conferencing. Suggested changed were understood during the meeting and same was accomplished using AutoCAD after the meeting. So, it was considered to be a waste of time and the officials at Australia wanted to get a solution where they can update the AutoCAD files on line. To provide a feasible solution of Lotus Domino was deployed. On top of this, ISM Sametime and Quickplace was also deployed. Quickplace was used to embed the AutoCAD file as object and Sametime was used as a conferencing media. During conferences, the associated AutoCAD file was put up in the whiteboard, changes were one during the meeting and the same was saved. So, embedding is a technology used in the practical field for real time updation of critical designs developed by AutoCAD.

MS Office OLE Automation Using C++

The following program will help in setting the font for the selected text in the active document.

NOTES

```
HRESULT CMSWord::SetFont(LPCTSTR szFontName, int nSize,
                     bool bBold, bool bItalic,COLORREF
crColor)
{
   if(!m_pWApp || !m_pActiveDocument) return E_FAIL;
    IDispatch *pDocApp;
    {
        VARIANT result;
        VariantInit(&result);
       OLEMethod(DISPATCH_PROPERTYGET, &result,
                 m_pActiveDocument, L"Application", 0);
        pDocApp= result.pdispVal;
    }
    IDispatch *pSelection;
    {
        VARIANT result;
        VariantInit(&result);
       OLEMethod(DISPATCH_PROPERTYGET, &result,
                  pDocApp, L"Selection", 0);
       pSelection=result.pdispVal;
    }
    IDispatch *pFont;
    {
        VARIANT result;
        VariantInit(&result);
       OLEMethod(DISPATCH_PROPERTYGET, &result,
                  pSelection, L"Font", 0);
        pFont=result.pdispVal;
    }
    {
       COleVariant oleName(szFontName);
      m_hr=OLEMethod(DISPATCH_PROPERTYPUT, NULL, pFont,
                      L"Name", 1, oleName.Detach());
        VARIANT x;
        x.vt = VT_I4;
        x.lVal = nSize;
      m_hr=OLEMethod(DISPATCH_PROPERTYPUT, NULL, pFont,
L"Size", 1, x);
        x.lVal = crColor;
      m_hr=OLEMethod(DISPATCH_PROPERTYPUT, NULL, pFont,
L"Color", 1, x);
        x.lVal = bBold?1:0;
      m_hr=OLEMethod(DISPATCH_PROPERTYPUT, NULL, pFont,
L"Bold", 1, x);
```

```
        x.lVal = bItalic?1:0;
       m_hr=OLEMethod(DISPATCH_PROPERTYPUT, NULL, pFont,
L"Italic", 1, x);
    }
   pFont->Release();
   pSelection->Release();
   pDocApp->Release();
   return m_hr;
}
```

NOTES

Check Your Progress

1. Define the term ActiveX control.
2. State about the ActiveX filtering.
3. What is object linking and embedding?
4. Write the main advantages of OLE.

4.3 ACTIVEX AND COM

ActiveX technology is an extension of OLE technology. ActiveX technology is basically categorized into six types of components.

- Automation Server
- Automation Controller
- Controls
- COM objects
- ActiveX Documents
- ActiveX Containers

Automation Server

Automation servers are components that can be derived from other applications. An automation server has one or more IDispatch Interface that can be implemented in other applications.

There can be three types of Automation servers.

1. Local: The local Automation server executes within its own space.
2. In-Process: The in-process automation server executes within the space of the controller.
3. Remote: The remote automation server executes on different machines from the controller.

An automation server can be implanted as DLL or EXE. The DLL can execute as local, in-process and as remote, but an EXE known as COM EXE can only execute as local or remote.

NOTES

Automation Controllers

Automation controllers are applications to access and manipulate the automation servers. An automation controller can be implemented as both EXE and DLL and can access all types of automation servers, local, in-process and remote.

The registry entries for the server and the controller decide which space the automation server will use in relation to the controller.

ActiveX Controls

An ActiveX control is a 2-bit control called OLE control or OCX. A typical ActiveX control has a User Interface both at the design time and the run time. An ActiveX control has a single IDispatch interface which defines the properties and methods of the control. An ActiveX control also implements an Interface IConnectionObject which consists of the events that can be fired by the control. An ActiveX control has many COM interfaces that the container application must support to use the features of the ActiveX control. An ActiveX control is always implemented as an In-Process to the container application in which it resides. Extension of the ActiveX control is OCX but it is simply known as Standard Windows DLL.

COM Objects

In architecture, a COM object is similar to an automation server and an automation controller COM object supports very less or no user interface. Every COM object is implemented in a COM interface. The automation controller which is going to access a COM object must have knowledge about the COM interface. The COM objects are used to implement the related data and corresponding functions.

ActiveX Documents

Microsoft Word and Microsoft Excel are examples of ActiveX documents servers and Internet Explorer is the example of ActiveX document controller. The ActiveX document is more than a control or an automation server. The ActiveX document architecture is the extension of Object Linking and Embedding. In this architecture, the document has more control over the container application in which the document control is hosted.

An ActiveX document is used within a uniform presentation architecture rather than in an embedded document architecture. The Internet Explorer is the perfect example of an active document container, which displays a web page, but the layout of the web page is dependent on the user's choice.

ActiveX Containers

An automation server is hosted by an ActiveX container application. A container application has to be robust to handle the cases in which the control lacks to provide an interface.

Component Object Model

COM is a standard communication protocol that allows objects to communicate through a special interface. This technology provides the facility to access objects and services outside the application boundary. It provides object-oriented solution for building and maintaining objects and services. It allows the use of binary

components rather than developing the components within the application. It provides the following functionalities to support the component development.

- Programming language independence: COM object can be developed in any language that supports the COM object layout.
- Reusability: COM objects are reusable because they are encapsulated and implemented in an isolated interface.
- Interoperability: COM provides an interoperability between the application and the object. More than one application can use the same object.
- Location transparency: COM objects are location independent because the COM objects can be implanted on the same machine on which they are used or can be implanted on different machines.
- Deployment: COM objects are self-contained components that make it very easy to deploy and use them.
- Efficiency: COM objects are small, light, fast and easy to use and reuse. They can be deployed anywhere.

COM Object

COM object can be created in any language and can be used in a language-independent environment. It can be created in any language that support its binary layout. COM objects are also independent from the machine on which they are created. They can be reused irrespective of their internal implementation as it forces a well-defined interface separate from the implementation.

A COM object concerns itself with how it interfaces with other objects. When such an object is not used in the environment in which it is created, an interface is exposed that can be seen from the outside environment. The COM object is a binary object, that is why it is machine independent. The application does not require the host environment or any interacting object to interact with the COM object.

COM object has three basic components.

- Class: The Class for a COM object can be created in any language. To use this class in a language-independent environment, it needs to be registered with the operating system.
- Object: Many instances of the COM object can be created using the function OnCreateInstance().
- Interface: Interface is a collection of class methods. Interface methods are used to update the class data.

COM Interfaces

COM interfaces consist of logically related well-defined methods that use known parameters and return types. An interface consists of only methods. There is no way to access the data within an interface except by using its methods. These methods must reside in an executable file or in a dynamic link library.

There are some rules for accessing objects through interfaces and these rules specify that the component must be logged in the system's registry. The rules

NOTES

also specify how the components are loaded and unloaded from the memory. These specifications are called the Component Object Model (COM).

The interface is the key which defines the object's behaviour irrespective of the operating system, hardware, programming language or version.

COM makes a strong distinction between an interface's definition and its implementation

An interface is defined using the Interface Definition Language (IDL) that is valid for any type of hardware, operating system and programming language.

The implementation of an interface on the other hand is language specific. The implementation in different platforms and languages works in the same way.

Example : A simple interface to implement the Employee COM object

```
Interface Employee: IUnknown
{
    public :
        virtual void –stdcall joins ( ) = 0;
        virtual void  - stdcall  resigns ( ) = 0;
}
```

Members of the interface must be public because the interface implementation is exposed to the world. COM interface tells everything about what a COM object does but it does not tell anything how it does all this.

It is possible that the deferent COM objects implement the same interface. A single method that an interface defines can yield different behaviour when invoked on the implementation supplied by the different objects. The one-to-one relationship between an interface and the various COM objects makes it possible for COM to offer a polymorphic behaviour. As long as the client program uses interfaces to access objects, the program may switch between different object implementations with a minimal impact on its code.

Accessing COM Object

COM objects are made available from the server that implements the interface. The client that uses the COM component must get a pointer to an interface to access the COM object. A COM object must implement at least one interface, although a COM object can implement as many interfaces as it may require.

A client can test for a particular behaviour and degrade gracefully if the object does not support the functionality. To test for a behaviour, the client must query an object at runtime to see whether the object supports a particular interface. COM also supports categories, or a collection of interfaces. If the object belongs to a particular category, it implements all the interfaces defined in that category.

To provide access to the client program , the object creates an array of function pointers called a vTable (The v stands for virtual) and passes a vTable pointer to the client. The client uses the vTable as the interface and uses the vTable pointer to locate a particular function pointer vptr . Once the client has the function pointer vptr, it can invoke the method directly. As long as the client program uses vptr to access a particular method , the changes in the method are invisible to the client program.

IUnknown Interface

The IUnknown interface is the base interface for all the other COM interfaces. This interface is used by OLE, ActiveX and other COM based applications. The IUnknown interface contains three methods that allow to get the pointer to an objects supported interface and to manage the object's interface pointer(s).

```
DECLARE_INTERFACE?(IUnknown)
{
  HRESULT QueryInterface(REFID riid, void ** ppvobj);
   Unsigned long Addref ();
   Unsigned long Release ();
};
```

To get a pointer to a COM interface, the IUnknown::QueryInterface() method is called, which returns a pointer to a specified interface for a particular object in its second parameter.

The first parameter specifies the ID of the interface being queried. This method lets the client move between the different interfaces the object implements. The client can switch to a different interface.

COM objects use reference counting to determine when an interface implementation can be freed. The `Addref()` and `Release()` methods keep track of an object's reference count. If the reference count is zero, the interface implementation is freed.

Interface Definition Language

An interface Definition Language (IDL) is also termed as interface description language. It is a specific language used for describing the interfaces between the various software components. It describes an interface in a language-neutral method to enable interaction between software components. The Microsoft Interface Definition Language (MIDL) describes interfaces between client and server programs. MIDL supports all client/server applications. An interface definition written in IDL has the following structure:

```
<interface> ::= <interface_header> { <interface_body> }
```

The COM standard is language independent both at defining the COM object and defining the COM client, but there must be some official language for defining the interfaces and the COM class. COM uses an Interface Definition Language (IDL) for defining interfaces. IDL is similar to C but provides object oriented extension. Visual C++ uses IDL to develop COM-based projects. When a COM-based project is compiled, the language compiler directs the IDL file to the Microsoft Interface Definition Language (MIDL) compiler. The MIDL compiler produces a binary description file called a type library.

COM Identifiers

A COM object is accessed through the use of a Universly Unique Identifier (UUID).

GUID: Each COM object is uniquely identified by a Global Unique Identifier (GUID). GUIDs are 8-byte numbers generated by a tool GUIDGEN. Each COM class has two IDs, one for the class and the other for the Interface.

NOTES

The CLSID is registered in the Windows registry which contains the path where DLL or EXE containing the corresponding class can be found. The CLSID can be found in the Windows Registry under the path HKEY_CLASSES_ROOT\CLSID.

The IID is used to invoke the methods of the class. The IID is also registered in the Windows registry. The IID can be found in the Windows Registry under the path HKEY_CLASSES_ROOT\Interface.

COM Runtime Library

A programming language compiler uses runtime library to implement built-in function during execution of the program. This special library usually uses functions for input, output and memory management. The concept of a runtime library should not be confused with an ordinary program library like that created by an application programmer or delivered by a third party or a dynamic library, meaning a program library linked at run-time.

COM builds a runtime library to provide services, functions and interfaces to support it. Whenever a COM component is written, all its functions and interfaces are available in runtime library. It also provides the services necessary to access COM component across the processes.

A running instance of a component can be obtained by asking the COM runtime library for a class factory and specifying the CLSID of the required component. The returned class factory is then asked to create an instance of the desired component, specifying an IID to obtain a particular interface that the component implements. A client then calls interface methods, and finally releases the interface by calling the Release () method.

Each process that uses COM in any way—client, server, object implementor—is responsible for three things:

i. Verify that the COM Library is a compatible version with the COM function CoBuildVersion.

ii. Initialize the COM Library before using any other functions in it by calling the COM function CoInitialize.

iii. Un-initialize the COM Library when it is no longer in use by calling the COM function CoUninitialize.

The COM Library may implement other functions to support persistent storage, naming, and data transfer without the "Co" prefix. The COM Library provides an implementation of a memory allocator. Whenever ownership of an allocated chunk of memory is passed through a COM interface or between a client and the COM library, this allocator must be used to allocate the memory.

Marshalling

When one COM process (COM client) needs to communicate with the other COM process (the server), it cannot simply access the data because the COM component is accessed through a pointer to the COM interface which points to the some location in the server memory. Since the address spaces of the client and the server are different, the client cannot access the data in the server memory.

Marshalling is the process of copying a structure, referenced by the pointer, over to the other process address space (Figure 4.6).

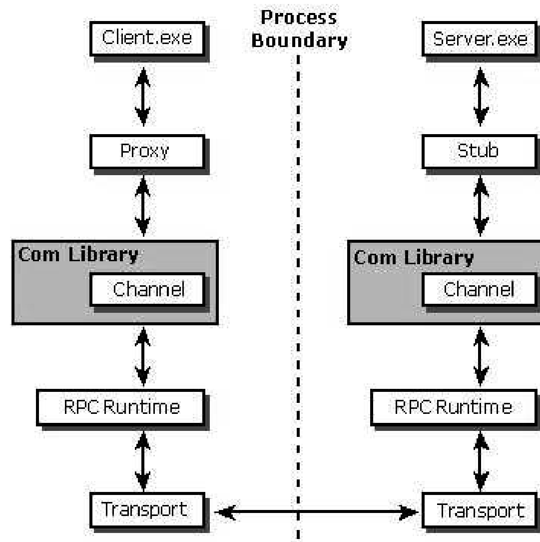


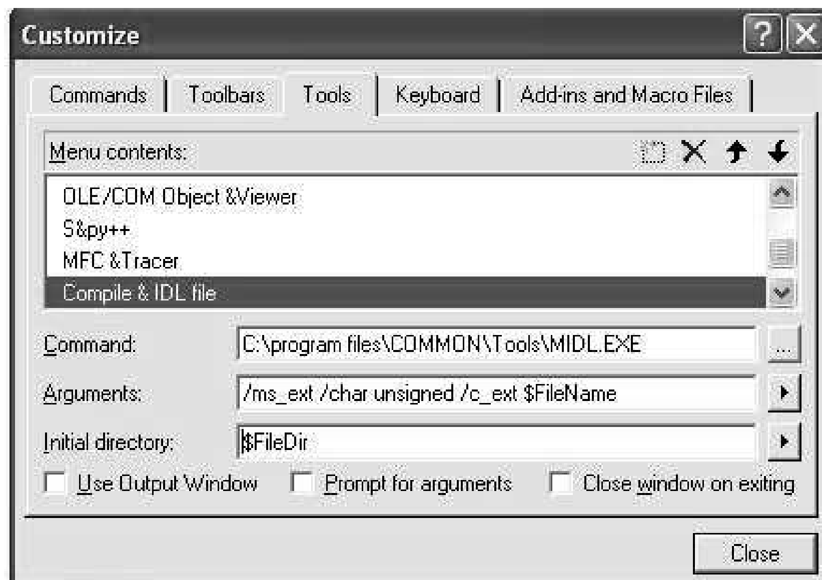
***Fig. 4.6** Marshalling Process*

Building COM Object

Building COM objects require the addition of some tools.

(a) MIDL Compiler

The MIDL compiler has become a standard component of the Visual C++ environment. It is used to compile the source code into the C code which is then compiled into a project by the Visual C++ compiler. Adding the MIDL compiler to the IDE allows easy compilation of the Interface Definition Language file.



To add MIDL compiler to Visual C++ environment:

- Select Customize command from the Tool menu.
- In the Command edit box, type the path of MIDL.EXE file.
- In the menu contents list box, type the Compile & IDL.file.

- In the Arguments text box, type /ms_ext /char unsigned /c_ext $FileName.
- In the Initial directory text box, type $FileDir.
- Click the check box

NOTES

- Use Output window.

(b) Registration Editor

The Registration editor is used to modify an operating system and the application settings. To add the registry editor to the Visual C++ environment:

- Select Customize command from the Tool menu.
- In the Command edit box, type path of REGEDIT.EXE file.
- In the menu contents list box, type Compile & Registry Editor.
- Empty the Arguments text box.
- In the Initial directory text box, type $FileDir.

(c) GUIDGEN

The GUIDGEN tool is used to generate a Global Unique Identifier for Interface and Class.

To add GUIDGEN to Visual C++ environment:

- Select Customize command from the Tool menu.
- In the Command edit box, type path of GUIDGEN.EXE file.
- In the menu contents list box, type & Generate New UUID.
- Empty the Arguments text box.
- Empty the Initial directory text box.

When building new COM components, defining the *custom* interfaces is required. A custom interface consists of a set of functions that are specific to the new component being developed. A custom interface is not already supported by the operating system. For example, a calendar component may contain a custom interface that contains a set of functions used by a program that uses the spell-checker component. Custom interface is defined separately from client and server applications because the custom interface definition is needed to be shared amongst the client and server applications.

COM Server and Client

COM uses a server and a client to implement its features through the interface. The COM server is a component that resides in the EXE or DLL file and provides services to the COM client. The COM server and client communicate through the COM interface.

The COM client gets a pointer to the COM server and uses the COM objects implanted on the server through the COM interface. COM object is available to all COM client applications, which get access to the pointer to COM Interface implemented on the server.

The COM server is implemented in two ways: in-process server and out-of-process server. The in-process server means both the COM server, which creates the COM object, and the client application. The Out-of-Process COM server means the COM server and the COM client that may be available on the same machine or different machines. The in-process server is implemented as a Dynamic Link Library (DLL) and out-of-process server is implemented as an executable file (.EXE). COM provides the mechanism to run the in-process server in a surrogate EXE process to allow other remote clients to access the COM object available on the server (Figure 4.7).

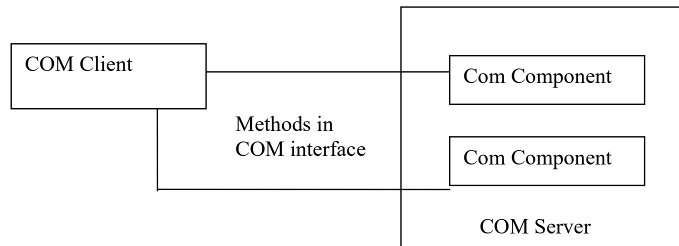
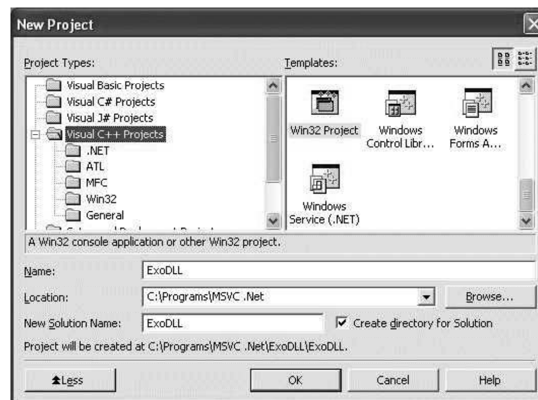


Fig. 4.7 *COM Server and Client*

4.3.1 Creating MFC Project to Develop Car Component

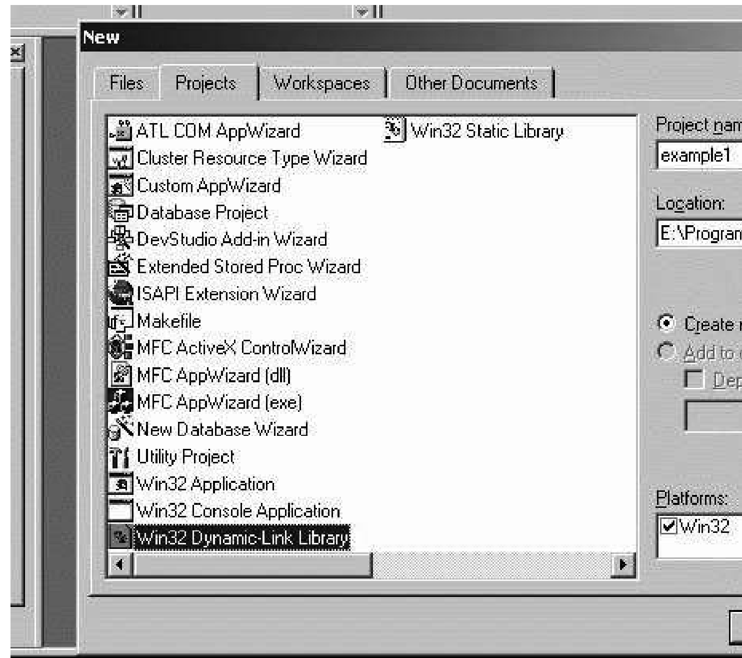
The Car component contains a COM interface ICar. The project is implemented as DLL, implementing project as DLL will create an in-process server. The DLL file does not contain any MFC code; code is created by MIDL compiler.

- Select New command from File menu. This will open the New dialog. This is the starting point for creating all kinds of projects.
- Select the project tab from new dialog box. In the New Project dialog box, in the Project Types list, click the Visual C++.



- Select DLL in project type and insert project name.

NOTES



Create Custom Interface Definition File

Once the interface is defined, the MIDL compiler creates the code for parameter marshaling even though it is not required in the In-process server.

Interface definition file ICar.IDL in car component project is:

```
[
object,
uuid(0132B310-5AC0-11d0-908D-00A0E71FAE43),
pointer_default(unique)
]
interface ICar : IUnknown
{
import "unknwn.idl";
HRESULT StartCar([out] BOOL *cCar);
HRESULT Accelrate(); }
```

UUID is the universal unique ID of the object and is created by GUIDGEN utility.

To generate unique ID in Visual C++ environment:

- Select Generate UID command from the Tool menu.
- Select GUID dialog box to generate the unique identifier for COM interface.
- Select registry format for the UID.
- Copy GUID and paste it in the Interface Definition file.

Since IUnknown is the standard interface supported by the operating system, it is only needed to import in the Interface Definition file. ID file includes the functions supported by the ICar component.

NOTES

Compiling IDL File

To compile an IDL file :

- The File is to be loaded in Visual C++ environment.
- Select the Compile command from the Tool menu to compile an IDL file.

After compilation, the MIDL compiler generates the source code (C like code) to support this interface.

MIDL compiler generates the following file after compiling IDL file:

- ICar.H: Header file for interface
- ICar_I.C: IDL file shared between server and client.
- ICar_P.C: Proxy code that implements a marshalling code for the interface.
- DLLDATA.C: Reference file for loading the correct interface.

All these files are added in the ICar project. Select Insert file into project command to insert all these files.

Creating Definition File ICar.DLL

A Library Definition File (DEF) is created for DLL to define which functions are exported by DLL.

```
ICar.DEF file
LIBRARY ICAR
DESCRIPTION 'ICAR Interface Marshaling'
EXPORTS
DllGetClassObject
DllCanUnloadNow
```

DllGetClassObject and DllCanUnloadNow function are entry points.

The first statement in the file must be the LIBRARY statement. This statement identifies the .def file as belonging to a DLL. The LIBRARY statement is followed by the name of the DLL. The linker places this name in the DLL's import library.

The EXPORTS statement lists the names and, optionally, the ordinal values of the functions exported by the DLL. The DllGetClassObject retrieves the class object from a DLL object handler or object application. OLE does not provide this function. DLLs that support the OLE Component Object Model (COM) must implement DllGetClassObject in OLE object handlers or DLL applications.

The DllCanUnloadNow Function Determines whether the DLL that implements this function is in use. If not, the caller can unload the DLL from memory. OLE does not provide this function. DLLs that support the OLE Component Object Model (COM) should implement and export DllCanUnloadNow.

Adding RPC Library to the Project

Four RPC libraries must be included in the interface project, rpcndr.lib, rpcdce4.lib, rpcns4.lib and rpcrt4.lib. The ICar project has a file called RPCHELP.C that contains the necessary compiler programs for RPC support. The file RPCHELP.C must be added to the ICar project in order for the project to link properly.

The RPCHELP.C file can be added to the interface project using the Visual C++ development environment.

- Add to Project command to add file.

NOTES

The Interface Definition for ICar can be built to generate the ICar.DLL file.

- The interface must be registered in the Windows Registry. A ICAr.REg registration file is created manually to register the Interface.

```
HKEY_CLASSES_ROOT\Interface\{0132B310-5AC0-11d0-908D-00A0E71FAE43}
HKEY_CLASSES_ROOT\Interface\{0132B310-5AC0-11d0-908D-00A0E71FAE43} \ProxyStubClsid32
HKEY_CLASSES_ROOT\CLSID\{0132B310-5AC0-11d0-908D-00A0E71FAE43} = ICAR_PSFactory
HKEY_CLASSES_ROOT\CLSID\{0132B310-5AC0-11d0-908D-00A0E71FAE43}\InprocServer32 =
c:\prj\ICar\debug\icar.dll
```

To import the Icar.REG file to the Windows registry:

- Select the Registry Editor Command from the Tool menu.
- Select the Import Registry file command from the file menu in the Registry Editor.
- Select ICar.REG file and press OK.

Implementing COM Interface

After creating and registering the Interface, it should be implemented to use the COM object. The COM object can be implemented in both DLL and EXE. The object implemented in DLL is in-process COM object and the object implemented in EXE file is out-of-process COM object.

The client that uses the COM objects does not have difference in using in-process COM object and out-of-process COM objects. The in-process COM object is faster because parameter marshalling is not required.

The Component Object Model (COM) is basically a component software architecture which permits applications and systems to be built from software components. In COM, applications communicate with each other using the functions defined in system called interfaces. A COM interface is a powerfully-typed agreement between software components which provides unique set of semantically related operations or methods. Hence, an interface is the definition of predictable actions and expected tasks, for example, drag-and-drop support implementation. All the functionality which a component must implement for a drop target is composed into the IDropTarget interface and all the drag source functionality is in the IDragSource interface. The name of an interface starts with "I" by convention.

Creating COM Object

COM objects are created in DLL (in-process) files to implement the COM interface. Follow these steps to create objects:

NOTES

- Select New command from the File menu. This will display New Dialog box and then click the Projects tab.
- From Project tab, specify the Project Name, Location, Workspace, Dependency and Platforms options and then double-click the MFC AppWizard (DLL) icon. App Wizard builds a .DLL which does not do anything. The new .DLL will compile, but since it does not export any classes or functions yet, it is still essentially useless. To use this DLL add functionality to the DLL and modify client application to use DLL.
- On the General tab of the Project Settings dialog box, select regular DLL shared or static in MFC AppWizard dialog box.
- Select Automation option. This option causes the AppWizard to insert start-up and exit code for COM object.
- Select Finish.

Visual C++ supports three different DLL development scenarios:

i. **Building a regular DLL that statically links MFC:** A regular DLL statically linked to MFC is a DLL that uses MFC internally, and the exported functions in the DLL can be called by either MFC or non-MFC executables.

ii. **Building a regular DLL that dynamically links MFC:** A regular DLL dynamically linked to MFC is a DLL that uses MFC internally, and the exported functions in the DLL can be called by either MFC or non-MFC executables.

iii. **Building an MFC extension DLL:** These always dynamically link MFC. An MFC extension DLL is a DLL that typically implements reusable classes derived from existing Microsoft Foundation Class Library classes. Extension DLLs are built using the dynamic-link library version of MFC (also known as the shared version of MFC).

Accessing COM Object

The DLLGetClassOBject ( ) and DllCanUnloadNow ( ) are two entry functions to access in-process COM objects. These functions are not required in out-of-process COM objects. To access COM objects, these two functions must be exported from the DLL file.

The COM support functions are defined in MFC but are implemented in the server DLL. These support functions are also exported through the definition file (.DEF) of the DLL that uses the functions.

Icar.def : Declares the module parameters for the DLL.

```
LIBRARY “ICAR”
DESCRIPTION ‘FISH Windows Dynamic Link Library’
EXPORTS
DllCanUnloadNow PRIVATE
DllGetClassObject PRIVATE
DllRegisterServer PRIVATE
```

These three functions are automatically included in the MFC applications because the Automation option is on. If the Automation option is not selected then these functions should be implemented manually.

NOTES

(1) `DllGetClassObject (0)`

When a user requests a given COM object, the Component Object Library looks into the Windows registry for the `InProcServer` of the given `CLSID`. The DLL that implements the COM Object is then loaded into the memory, and the function `DllGetClassObject` is called. The `CLSID` of the COM Object implementing the interface and `IID` of the interface that the user is requesting are passed into the function. The DLL containing the COM Object then creates the appropriate class factory for the `CLSID` and returns the corresponding interface pointer for the `IID`. Since a `CLSID` is passed into `DllGetClassObject`, a DLL can contain many different COM Objects. The interface pointer is returned to the caller through the parameter ppv.

HRESULT DllGetClassObject (REFCLSID rclsid, REFIID riid, LPVOID *ppv)

MFC AppWizard inserts the following code in the DLL application for implementing the DllGetClassObject () code.

```
STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
LPVOID* ppv)
{
AFX_MANAGE_STATE(AfxGetStaticModuleState());
return AfxDllGetClassObject(rclsid, riid, ppv);
}
```

(2) `DllCanUnloadNow ( )`

Since the client application using the COM object does not link directly with the server creating the COM Object, an unload mechanism must be in place so that the unneeded COM Objects are removed from the memory. With in-process servers, this mechanism is through the function DllCanUnloadNow. The Component Object Library periodically asks each COM Object server if it can be unloaded from its memory. If the server returns to S_OK, the server DLL is removed from the memory. A COM Object server returns to S_OK when the clients have finished using the COM Objects.

STDAPI DllCanUnloadNow(void)

MFC AppWizard inserts the following code in the DLL application for implementing the DllCanUnloadNow () code.

```
STDAPI DllCanUnloadNow(void)
{
AFX_MANAGE_STATE(AfxGetStaticModuleState());
return AfxDllCanUnloadNow();
}
```

(3) `DllRegisterServer ( )`

The function DllRegisterServer is a useful method for having the COM Object server correctly register with the Windows registry. When this function is called, the COM Object server updates the Windows registry with the settings necessary for client applications to use the server.

NOTES

BOOL DllRegisterServer(void)

MFC AppWizard inserts the following code in the DLL application for implementing the DllRegisterServer () code.

```
// by exporting DllRegisterServer, you can use regsvr.exe
STDAPI DllRegisterServer(void)
{
AFX_MANAGE_STATE(AfxGetStaticModuleState());
COleObjectFactory::UpdateRegistryAll();
return S_OK;
}
```

Creating Class in ICAR.DLL

A COM class is created to implement the COM object using the ClassWizard.

- Select the ClassWizard command from the View menu in the Visual C++ environment.
- A new class CCar is created from the New Class dialog box.
- Select CCmdTarget class as the base class of the CCar class.

The CCmdTraget class is chosen as the base because it implements IUnkown interface which is the base Interface for all the custom COM interfaces. This also allows to create the COM object from the MFC class COleObjectFactory. The COleObjectFactory is called from COM entry point DllGetClassObject, otherwise a special code is needed to create the COM object.

The COM class CCar defines the method to enable the MFC Interface map to call routines to implement the IUnknown interface. These routines are called by a number of macros defined in the COMMARCOS.H file. The COMMARCROS.H file is not part of the MFC and must be included in the project.

```
#ifndef _COMMACROS_H
#define _COMMACROS_H
#ifndef IMPLEMENT_IUNKNOWN
#define  IMPLEMENT_IUNKNOWN_ADDREF(ObjectClass,
InterfaceClass)\
STDMETHODIMP_(ULONG)ObjectClass::X##InterfaceClass::AddRef(void)\
{ \
METHOD_PROLOGUE(ObjectClass, InterfaceClass); \
return pThis->ExternalAddRef(); \
}
#define  IMPLEMENT_IUNKNOWN_RELEASE(ObjectClass,
```

NOTES

```
InterfaceClass)\
STDMETHODIMP_(ULONG)ObjectClass::X##InterfaceClass::Release(void)\
{ \
METHOD_PROLOGUE(ObjectClass, InterfaceClass); \
return pThis->ExternalRelease(); \
}
#define IMPLEMENT_IUNKNOWN_QUERYINTERFACE(ObjectClass,
InterfaceClass)\
S T D M E T H O D I M P
ObjectClass::X##InterfaceClass::QueryInterface(REFIID
riid, LPVOID *pVoid)\
{ \
METHOD_PROLOGUE(ObjectClass, InterfaceClass); \
return (HRESULT)pThis->ExternalQueryInterface(&riid
,ppVoid); \
}
#define IMPLEMENT_IUNKNOWN(ObjectClass, InterfaceClass)\
IMPLEMENT_IUNKNOWN_ADDREF(ObjectClass, InterfaceClass)\
IMPLEMENT_IUNKNOWN_RELEASE(ObjectClass, InterfaceClass)\
IMPLEMENT_IUNKNOWN_QUERYINTERFACE(ObjectClass,
InterfaceClass)
#endif #endif
```

Creating Unique Class ID

The class implementing the COM interface must be identified by a unique CLID. The CLID is created using GUIDGEN utility or UUIDGEN program. The UUIDGEN program is a command-Line program, whereas GUIDGEN utility is a GUI based program. To obtain single GUID, the UUIDGEN program is executed on command-line with no argument. Output of the UUIDGEN program can be redirected to the text file. To directly put the output of this utility to the file, –o argument is used.

```
C: \UUIDGEN –o NewGID.txt
```

This command copies one GUID in file NewGID.txt. To generate bunch of GUIDs, the UUIDGEN utility is executed with –n argument.

```
C:\ UUIDGEN –n 10
```

This command will generate 10 GUIDs. The GUIDGEN utility serves exactly the same purpose as UUIDGEN program. GUIDGEn does not generate more than single GUID at once. The GUIDGEN utility does have more options than UUIDGEN program as it can generate GUID in the format that supports IMPLEMENT_OLECREATE macro that MFC uses to declare GUIDs.

Header File for Implementing CICar Class

```
#ifndef _CLSID_CICar
#define _CLSID_CICar
//{AFA853E0-5B50-11d0-ABE6-D07900C10000}
```

```
DEFINE_GUID(CLSID_CICar,0xAFA853E0,0x5B50,0x11d0,
0xAB,0xE6,0xD0,0x79,0x00,0xC1,0x00,0x00); #endif
```

The DEFINE_GUID macro connects the name CLSID_CICar to the CLID (Class ID) generated by GUIDGEN utility.

NOTES

Check Your Progress

5. What are automation controllers?
6. Define a Component Object Model (COM).
7. What is Interface Definition Language (IDL)?
8. Define an MIDL compiler.
9. What is a COM server?
10. Name the four RPC libraries that must be included in the interface.
11. Which is faster, the in-process COM object or the out-of-process COM object?

4.4 ACTIVEX CONTROL MACROS

ActiveX controls are used extensively in the worksheet forms. It can be used with or without VBA code. It is also used on VBA UserForms. Normally, ActiveX controls are used to design flexible solutions than those which are catered by the Form controls. ActiveX controls consists of widespread attributes which is used to customize its appearance, associated behaviour, fonts, and other features.

Various events occurs when ActiveX controls are interacted. These events can completely be controlled using ActiveX. For example, when a user selects an option, its associated actions can be controlled. Complete control can be set when a database is asked for a specific set of data to fill up a combo box when a button is clicked by a user. Macros can also be written using ActiveX which will answer to consequences in conjunction with ActiveX controls. A desktop or laptop also consists of various ActiveX controls which were deployed during the installation of MS Office e.g., Calendar Control, Windows Media Player etc.

ActiveX controls are extensively used to support web applications for years.

Interaction between Controls Windows and ActiveX Control Containers

In case, a control is handled within a control container, it uses two processes for communication. They are as follows:

- The process disclose its corresponding properties and methods.
- The process triggers the associated events.

Figure 4.8 portrays the process of implementation of the procedures.

NOTES

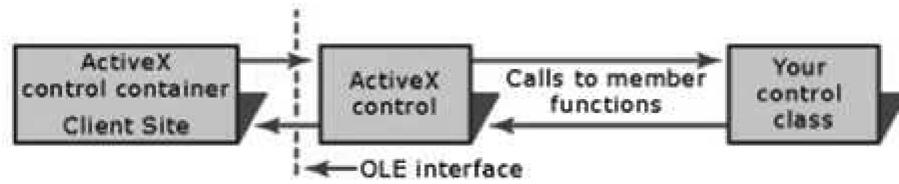


Fig. 4.8 *Communication between an ActiveX Control Container and an ActiveX Control*

At the same time, Fig. 4.8 also demonstrates the process of how the supplementary OLE makes communications and (apart from automation and events) addressed by the controls.

The communication of the controllers with the containers are achieved by COleControl. To address the requests of certain container, COleControl will request member functions which are deployed within the control class. All methods and certain properties are addressed in this similar mode. The class of the control can also trigger interaction with the associated container by requesting the member functions of COleControl.

Active and Inactive States of an ActiveX Control

A control primarily consists of two basic states. They are active state and inactive state. Usually, these states are identified by the control consists of a window. An active control certainly has a window where as an inactive control do not have any window. With the initiation of windowless activation, this kind of dissimilarity is no longer supported, but this concept is still applicable to various controls.

When a windowless control becomes active, it invokes mouse capture, keyboard focus, scrolling, and other window services from its container. You can also provide mouse interaction to inactive controls, as well as create controls that wait until activated to create a window.

When a control with a window is in active state, it can communicate completely with the control container, the associated users and Windows. Figure 4.9 portrays the trails of interaction between the ActiveX control, the control container and the operating system.

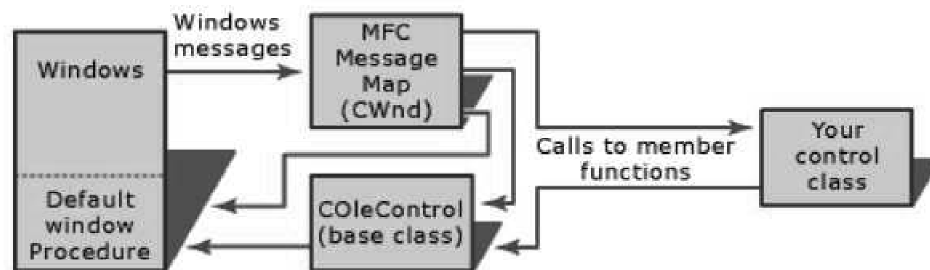


Fig .4.9 *Windows Message Processing within a Windowed ActiveX Control (When Active)*

NOTES

Serialization

Serialization is also termed as “Persistence”. It permits the control to write the value of its properties to “Persistent Storage”. Controls then can be recreated via understanding the object’s situation from the storage. In other words, serialization is the method of reading from or writing to a permanent storage medium, i.e., disk storage. The Microsoft Foundation Class (MFC) Library offers built-in assistance for serialization in class “CObject”. “COleControl” encompasses this support to ActiveX controls by means of property exchange mechanism.

“Serialization” for ActiveX controls is deployed by superseding COleControl::DoPropExchange. This function is invoked at the time of loading and saving the control object and is responsible for storing every property deployed with a “member variable” or a “member variable” with alteration announcement.

The code given below is appended to classes generated with the ActiveX Control Wizard.

```
void CMyAxUICtrl::DoPropExchange(CPropExchange* pPX)
{
  ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
  COleControl::DoPropExchange(pPX);

  // TODO: Call PX_ functions for each persistent custom
property.
}
```

If it is required to create a persistent property, DoPropExchange should be modified by appending a call to the “property exchange” function. The code given below illustrates the serialization of a custom “Boolean CircleShape” property, where the “CircleShape” property has a default TRUE value.

```
void CMyAxSerCtrl::DoPropExchange(CPropExchange* pPX)
{
  ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
  COleControl::DoPropExchange(pPX);
PX_Bool(pPX, _T("CircleShape"), m_bCircleShape, TRUE);
}
```

Installing Procedure of ActiveX Control Classes and Associated Tools

When you install Visual C++, the MFC ActiveX control classes and retail and debug ActiveX control run-time DLLs are automatically installed if ActiveX controls are selected in Setup (they are selected by default).

By default, the ActiveX control classes and tools are installed in the following subdirectories under \Program Files\Microsoft Visual Studio .NET.

- **\Common7\Tools :** Consists of the test container files (TstCon32.exe, long with its associated Help files).
- **\Vc7\atlmfc\include :** Consists of the include files which are required for developing the ActiveX controls with MFC.

- **\Vc7\atlmfc\src\mfc :** Consists of the source code for associated ActiveX control classes in MFC.
- **\Vc7\atlmfc\lib :** Consists of the libraries which are required for developing the ActiveX controls with MFC.

NOTES

An ActiveX control container is considered as parent program which provides the setting for an ActiveX control for execution.

- ActiveX control is a control which uses Microsoft ActiveX techniques.
- ActiveX is not at all a programming language. It is basically a set of rules which specifies the method of how various applications shares information between various applications.
- Programmers can create ActiveX controls using programming code in various languages e.g., C, C++, Visual Basic, and Java.
- An application which is proficient in consisting ActiveX controls with or without MFC can be developed at ease and this will be much simpler to code with the help of Microsoft Foundation Class Library (MFC).

When an ActiveX control is inserted into a specific project, the ClassWizard produces a C++ wrapper class. This is the resultant from CWnd and it is tailor made to the control's methodologies and associated attributes. The class consists of member functions for every properties and methods. It also consists of constructors which can be used dynamically for creating an occurrence of the specific control. ClassWizard also produces wrapper classes for objects which are utilized by the control.

Following code shows some specific member functions from the file Calendar.cpp which ClassWizard produces for the Calendar control.

```
unsigned long CCalendar::GetBackColor()
{
    unsigned long result;
    InvokeHelper(DISPID_BACKCOLOR, DISPATCH_PROPERTYGET,
VT_I4, (void*)&result, NULL);
    return result;
}

void CCalendar::SetBackColor(unsigned long newValue)
{
    static BYTE parms[ ] = VTS_I4;
    InvokeHelper(DISPID_BACKCOLOR, DISPATCH_PROPERTYPUT,
VT_EMPTY, NULL, parms, newValue);
}

short CCalendar::GetDay()
{
    short result;
     InvokeHelper(0x11,  DISPATCH_PROPERTYGET,  VT_I2,
(void*)&result, NULL);
```

```
    return result;
}

void CCalendar::SetDay(short nNewValue)
{
    static BYTE parms[ ] = VTS_I2;
    InvokeHelper(0x11, DISPATCH_PROPERTYPUT, VT_EMPTY,
NULL, parms, nNewValue);
}

COleFont CCalendar::GetDayFont()
{
    LPDISPATCH  pDispatch;
    InvokeHelper(0x1, DISPATCH_PROPERTYGET, VT_DISPATCH,
(void*)&pDispatch, NULL);
    return COleFont(pDispatch);
}

void CCalendar::SetDayFont(LPDISPATCH newValue)
{
    static BYTE parms[ ] = VTS_DISPATCH;
    InvokeHelper(0x1, DISPATCH_PROPERTYPUT, VT_EMPTY,
NULL, parms, newValue);
}

VARIANT CCalendar::GetValue()
{
    VARIANT result;
    InvokeHelper(0xc, DISPATCH_PROPERTYGET, VT_VARIANT,
(void*)&result, NULL);
    return result;
}

void CCalendar::SetValue(const VARIANT& newValue)
{
    static  BYTE  parms[ ] = VTS_VARIANT;
    InvokeHelper(0xc, DISPATCH_PROPERTYPUT, VT_EMPTY,
NULL, parms, &newValue);
}

void CCalendar::NextDay()
{
    InvokeHelper(0x16, DISPATCH_METHOD, VT_EMPTY, NULL,
NULL);
}
```

NOTES

```
void CCalendar::NextMonth()
{
    InvokeHelper(0x17, DISPATCH_METHOD, VT_EMPTY, NULL,
NULL);
```

NOTES

Advantage of ActiveX Control

- ActiveX controls can be customized as per requirement and can also be created without any challenges.
- Flexibility in designing is possible during the development of ActiveX Control.
- ActiveX controls are used to enhance the power of programs and scripts.
- ActiveX controls are loaded separately in the memory. Hence, these program gets executed in an easily fashion.
- Flow of ActiveX control can be done easily using appropriate programming skills.
- Numerous controls can be activated.
- ActiveX controls are very common things and is utilized on a daily basis by various applications.
- ActiveX control save a huge amount of time.

Disadvantages of ActiveX Control

- ActiveX controls are only meant for Windows environment. They can't be implemented in Linux or MAC. Due to this type of rigidity, many users do not prefer to use it.
- ActiveX controls are never used for simple scripts.
- Enabling ActiveX controls in the system might lead to security vulnerabilities.
- By default, ActiveX controls are disable.

Signed ActiveX Controls

A signed ActiveX control specifies that the specific control is authentic and is certified by an established or identified authority and has never been modified since the authorization and authentication. Microsoft implemented the procedure of signing ActiveX controls for deploying specific volume of security to the components of the associated ActiveX controls.

4.5 BUILDING AN ACTIVEX SERVER APPLICATION

Process to create an ActiveX control as an Automation server

MFC ActiveX control can be developed as an Automation server. The concept is to embed controls into another application using developed programs. Also, programs can be developed to call control from another application. Such kind of controls has to be hosted in the ActiveX control container.

NOTES

Following steps are followed to create a control as an Automation server.

- **Create the control**: The ActiveX controls can be created using wizards provided my Microsoft. The MFC starter program consists of C++ source (.cpp) files, resource (.rc) files, and a project (.vcxproj) file. The code which are generated in the stated starter files are created on MFC.
- **Add methods**: An ActiveX control fire events for interacting between itself and its associated control container. Additionally, a container also interacts with a control via methods and appropriate properties. Methods are also known as functions. Methods and properties enable to transfer interface so that it can be used by other applications as well. For example, Automation clients and ActiveX control containers.
- **Override IsInvokeAllowed**: This function enables the invocation of the automation process.

Following steps are used for accessing the methods in an Automation server, using programs.

1. An application is created, e.g., MFC exe.
2. At the starting of the InitInstance function, the following line needs to be inserted:
   ```
   AfxOleInit();
   ```
3. In the "Class View", right-click on the "project node"" and select ""Add class" from typelib for importing the type library.

   This process will add the files with the file name and extensions .h and .cpp to the created project.
4. The following line needs to be added in the class's header file from where methods are to be called in the ActiveX control.
   ```
   #include filename.h
   ```
   <file name should be the name of the header file which was created at the time of importing the type library>
5. Now, let us consider the function from where a call should be initiated to a specific method in the ActiveX control, appropriate codes to be appended which will create an object of the control's wrapper class and subsequently the ActiveX object should be created. For example, the code below stated MFC code which reflects a CCirc control, receives the Caption property and exhibits the outcome when the OK button is clicked within a dialog box.

```
void CCircDlg::OnOK()
{
    UpdateData();    // Get the current data from the
dialog box.
    CCirc2 circ;     // Create a wrapper class for the
ActiveX object.
   COleException e;   // In case of errors
```

NOTES

```
// Create the ActiveX object.
// The name is the control's progid; look it up using
OleView
if (circ.CreateDispatch(_T("CIRC.CircCtrl.1"), &e))
{
  // get the Caption property of your ActiveX object
  // get the result into m_strCaption
  m_strCaption = circ.GetCaption();
  UpdateData(FALSE);   // Display the string in the
dialog box.
}
else { // An error
  TCHAR buf[255];
  e.GetErrorMessage(buf, sizeof(buf) / sizeof(TCHAR));
  AfxMessageBox(buf);   // Display the error message.
}
}
```

Responsibilities of a COM Server

A COM server is an object which provides service to an associated client. These services are always in the type of COM interface installed and this can be requested by any client which is capable of receiving a pointer to the interfaces on the server object. A COM client is a code or object which is able to establish a pointer to a COM server. COM client uses the services provided by the COM server calling the procedures of its associated interfaces. COM servers are only association of compiled code which is called by executing processes. The server should implement codes for a specific class object via installation of either the IClassFactory or IClassFactory2 interface.

There are two categories of servers which are as follows.

- In-process
- Out-of-process

In-process servers are deployed in a dynamic linked library i.e., DLL. Out-of-process servers are deployed in an executable file i.e., EXE. Out-of-process servers resides both on local or remote desktops. Moreover, COM delivers a process which enables an in-process server (a DLL) to execute in a substitute EXE process to achieve the benefit of being competent in executing the associated process on a remote desktop. It is taken into consideration that numerous COM servers are installed in DLL, but it may not always be correct. There are exceptions also like Word, Excel, etc.

Building a LOCAL COM Server

- **Step 1**: Start the VC6.0 application and create a new WIN32 Application workspace. Select "a simple application", and provide a suitable name. In this example, we will name it as CarLocalServer.

- **Step 2**: Create the idl file. This file will consist of the following code. A name CarLocalServerTypeInfo.idl is provided.

```
import "oaidl.idl";

// define IStats interface
[object, uuid(FE78387F-D150-4089-832C-BBF02402C872),
 oleautomation, helpstring("Get the status information
about this car")]
interface IStats : IUnknown
{
   HRESULT DisplayStats();
   HRESULT GetPetName([out,retval] BSTR* petName);
};

// define the IEngine interface
[object, uuid(E27972D8-717F-4516-A82D-B688DC70170C),
 oleautomation, helpstring("Rev your car and slow it down")]
interface IEngine : IUnknown
{
   HRESULT SpeedUp();
   HRESULT GetMaxSpeed([out,retval] int* maxSpeed);
   HRESULT GetCurSpeed([out,retval] int* curSpeed);
};

// define the ICreateMyCar interface
[object, uuid(5DD52389-B1A4-4fe7-B131-0F8EF73DD175),
 oleautomation, helpstring("This lets you create a car
object")]
interface ICreateMyCar : IUnknown
{
   HRESULT SetPetName([in]BSTR petName);
   HRESULT SetMaxSpeed([in] int maxSp);
};

// library statement
[uuid(957BF83F-EE5A-42eb-8CE5-6267011F0EF9), version(1.0),
 helpstring("Car server with typeLib")]
library CarLocalServerLib
{
   importlib("stdole32.tlb");
   [uuid(1D66CBA8-CCE2-4439-8596-82B47AA44E43)]
   coclass MyCar
   {
     [default] interface ICreateMyCar;
```

NOTES

```
        interface IStats;
        interface IEngine;
    };
};
```

NOTES

The above code should be appended into the project which is created. Also, it should be noted that all the Global Unique Identification numbers (GUID) must be produced via guidgen.exe, so that the created IDs will not be similar as stated in the above code.

- **Step 3**: The following files should be inserted in the appropriate workspace: CarLocalServer.cpp, MyCar.cpp, MyCarClassFactory.cpp, Locks.cpp, MyCar.h, MyCarClassFactory.h and Locks.h.

Immense attention should be paid while including the .h files. Always stdafx.h should be included first before including any other files and this is mandatory. In the absence of this, there is a possibility that the compiler will start generating error messages. Now, stdafx.h should be opened to see if the below mentioned codes exists inside the file:

```
#define WIN32_LEAN_AND_MEAN
// Exclude rarely-used stuff from Windows headers
```

If it exists, this needs to be removed or commented.

- **Step 4**: Now, the .reg file needs to be created. Any names can be used for this file but the content has to be exactly the same which is specified above. There are exception to this also which are as follows:
  - o Only the generated ID should be used.
  - o Path specified above can be replaced with the actual path.
  - o A different name can be used (the coclass name MyCar should not be altered).

After creating this file, the file should be double clicked and Windows system shall inform whether the obligatory components are registered effectively or not.

o **Step 5**: The entire project should be built with no errors.

GUID structure (guiddef.h)

GUIDs are Microsoft implementation of the Distributed Computing Environment (DCE) Universally Unique Identifier (UUID). A GUID recognizes an object as a COM interface or a COM class object or a manager Entry-Point Vector (EPV). GUID is 128-bit in length and comprises of a set of 8 hexadecimal digits, followed by 3 sets of 4 hexadecimal digits each, followed by 1 set of 12 hexadecimal digits. An example of GUID is: 8B37DC33-FB32-1762-C35A-44BB212551DB.

The GUID structure is as follows

```
typedef struct _GUID {
  unsigned long  Data1;
  unsigned short Data2;
  unsigned short Data3;
  unsigned char  Data4[8];
```

```
} GUID;
```

Where,

Data1 - Denotes the first 8 hexadecimal digits of the GUID.

Data2 - Denotes the first group of 4 hexadecimal digits.

Data3 - Denotes the second group of 4 hexadecimal digits.

Data4 – It is the array of 8 bytes. The first 2 bytes consists of the third group of 4 hexadecimal digits. The remaining 6 bytes consists of the concluding 12 hexadecimal digits.

NOTES

4.6 BUILDING ACTIVEX CONTROL

The ActiveX control can be built by clicking the Build button. This will generate the ActiveX control file (**.ocx**) and will automatically register the control on the system.

After registering the control, it can be accessed by the container program. Leave all the default settings and click the OK button to create the project.

It generates three classes

—Ctrl— derived from ColeControl

—App— derived from ColeControl Module.

—PropPage— derived from ColeProperty Page.

Visual C++ classWizard creates a ellipse to display the control. A frame to change the size of the control surrounds the ellipse.

Whenever the control needs drawing or redrawing, the OnDraw member function of the control class is called.

Example

- OLE controls are basically created using the resource editor or any code can be used to create the OLE control.

  Bitmaps are created with IDs, IDB-BITMAP1 and IDB-BITMAP2.

- Add declarations for handling the control.

  class — Ctrl: public COLEControl

```
{
    :
public :
Cbitmap m_CurrentBitmap, m_Bitmap N;
Cbitmap m-Bitmap M;
```

- Constructor or OnDraw functions can be used to initially display control.

```
——Ctrl : : ——Ctrl()

{
m_BitmapN. LoadBitmap ( IDB_BITMAP1);
m_BitmapN. LoadBitmap ( IDB_BITMAP2);
```

```
m_CurrentBitmap = &m_Bitmap N;
}
```

NOTES

```
void— Ctrl : : OnDraw ( CDC * pDC, Crect & reBoundas,
Const Crect & rcInvalid)

{

MITMAP BM;
CDC MemDC;
MemDC. Creat Comptible DC  (NULL);
MemDC SelectObject (*m_CurrentBitmap);
M_CurrentBitmap -> GetObject (Size of (BM),&BM);
PDC-> BitBlt
(( reBounds.right-BM.bmWidth)/2,
(reBounds.bottom-BM.bmHeight)/2,
BM.bmWidth'
BM.bmHeight,
&MemDC,
O,O,SCRCOPY);
}
```

Adding Message Handler

The ClassWizard is used to add the message handler function for control to modify it.

Example – When left mouse button is clicked and released pn OLE control, it should display

IDB_BITMAP2

```
void —Ctrl : : OnLButtonUp(UINT Flags,
(Point point)
{

if (m_CurrentBitmap = = &m_BitmapN)
m_CurrentBitmap - & m_Bitmap M;
else
m_CurrentBitmap - & m_BitmapN;
Invalidate Control ( );

}
```

Defining Properties

- Control can have two types of properties.
  (1) Stock: It is one of a set of common properties that MFC stores and initializes.

- The MFC code also performs the appropriate action when the value of a stock property is changed. Only the user has to enable the property and provide the code to use its value.
- To enable the property class, wizard's OLE Automation Tab is used. Select — Ctrl class from class list, click Add property button, in Add property dialog box, select name of the property from the external name list.
- The MFC code stores the value of the property.

The MFC also invalidates the control, thereby forcing the OnDraw function to redraw it whenever the value of the property changes.

(2) Custom Property: It is a property user device in itself, assigning it a name and providing most of the supporting code.

- To add the custom property in the Add property dialog box, name is defined in the External Name List, defines the type in Type text box, variable name is used to hold the value of the property and notification function is used to change the value.

Example:

- Stock property – Back color
- Custom Property – Show Frame.
- Custom Property needs to explicitly initialize; PX – functions are used for this.

void —— ctrl : : On prop exchange ((Prop exchange p^{px})

```
{

px Bool ( ppx , TC "ShowFrame), mShowFrame, False),
}
```

Container program can change the value of the custom property. Whenever the value is changed the notification function is called and the control needs to redraw.

- To redraw the control the invalidated control ( ) function is called in Notification function.

```
void  ——Ctrl :: OnShowFrameChanged()
{

InvalidateControl();
}
```

Modifying OnDraw() Function

To use current values of properties

```
void ——Ctrl:: OnDraw(CDC *pDC, const Crect & rcBound,
const Crect& rcInvalid)
{
Cgrush Brush(TranslateColor(GetBackColor());
PDC -> FillRect(rcBound, &Brush);
```

```
-
-
-
if(m_ShowFrame)
{
`
```

NOTES

ActiveX Source Code

```
(1)
// AXCtrl.h : main header file for AXCTRL.DLL
# if !defined ( AFX_AXCTRL_H_CC31D28C_B1B1_11D1_80FC_
00C0F6A83B7F_INCLUDED_ )
#defined AFX_AXCTRL_H_CC31D28C_B1B1_11D1_80FC_
00C0F6A83B7F_INCLUDED_
# if MSC_VER>1000
# pragma once
# endif // MSCVER>1000
# if !defined( AFXCTLH_ )
# error include 'afxctl.h' before including this file
# endif
# include "resource.h"        //main symbols

/////// ////////// //// //////// ////// //////// ///////
/// ////// ///// //// //////////
// CAXCtrlApp : See AXCtrl.cpp for implementation
class CAXCtrlApp : public C0leControlModule
{
public:
BOOL InitInstance( );
Int ExitInstance ( );
};
extern const GUID CDECL_tlid:
extern const WORD _wVerMajor;
extern const WORD _wVerMinor;

//{ AFX_INSERT_LOCATION }
// Microsoft Visual C++ will insert additional declaration
immediately before
// the previous line.
#        endif        //        !        defined
AFX_AXCTRL_H_CC31D28C_B1B1_11D1_80FC_00C0F6A83B7F_INCLUDED_)
______ ___ __ ___ ____ ___ ___ __ ____________________
(2)

// AXCtrl.cpp : Implementation of CAXCtrlApp and DLL
registration.
```

```
# include "stdafx.h"
# include "AXCtrl.h"# ifdef _DEBUG
# defined new DEBUG_NEW
# undef THIS_FILE
static char THIS_FILE [ ] = FILE;
# endif

CAXCtrlApp NEAR theApp;
Const GUID CDECL BASED_CODE _tlid =
{ 0xcc31d283, 0xb1b1,0xlidl' { 0x80,
oxfc,0,0xc0,0xf6,0xa8,0x3b'0x7f  }};
const WORD –wVerMajor = 1;
const WORD –wVerMinor = 0;

/////// ///////// //////// ///////// //// /////// //////
// ////// ///// /////// ///////
// CAXCtrlApp ::InitInstance ( )
{
Bool bInit  =  C0leControlModule :: InitInstance( );
If ( bInit )
{
// TODO: Add your own module initialization code here.
}
return bInit;
}

//////// //////// ///// //////// ////// ////////// /////
///// // /////////////
// CAXCtrlApp ::ExitInstance – DLL termination
int CAXCtrlApp::ExitInstance( )
{
// TODO : Add your own module  termination code here.
Return ColeControlModule ::ExitInstance ( );
}
/// ///////// /////// //////// /////// ///////// ///////
// //////// ///////////
// DllRegisterServer-Adds entries to the system registry

STDAPI DllRegisterServer( void )
{
AFX_MANAGE_STATE(_afxModuleAddrThis );

If ( !AfxOleRegisterTypeLib(AfxGetInstanceHandle( )' –
tlid) )
Return ResultFromScode (  SELFREG_E_TYPELIB );
```

NOTES

```
If ( !ColeObjectFactoryEx :: UpdateRegistryAll ( TRUE ) )
Return ResultFromScode ( SELFREG_E_CLASS );
Return NOERROR;
}

//////// ///// ////////// //////// //////////// ///////
/////// ///// ////////
// Dll UnregisterServer - Removes entries to the system
registry

STDAPI DllUnregisterServer( void )
{
AFX_MANAGE_STATE( _afxModuleAddrThis );

If ( !AfxOleUnregisterTypeLib ( _tlid, _wVerMajor,
_wVerMinor ) )
return ResultFromScode ( SELFREG_E_TYPELIB );

If ( !ColeObjectFactoryEx ::UpdateRegistryAll(FALSE ))
Return ResultFromScode ( SELFREG_E_CLASS );
Return NOERROR;
}

________ __ ________ ___ _____ _ ____
(3)

// AXCtrl.h : Declaration of the CAXCtrlCtrl ActiveX
Control class.

# if !defined (
AFX_AXCTRLCTL_H_CC31D294_B1B1_11D1_80FC_00C0F6A83B7F_INCLUDED_
)
# defined
AFX_AXCTRL_H_CC31D294_B1B1_11D1_80FC_00C0F6A83B7F_INCLUDED_

#  if MSC_VER>1000
# pragma once
# endif // MSCVER>1000
/////// /////// ////////////// ///////////////// /////
///// ////// /////
// CAXCtrlCtrl : See AXCtrlCtl.cpp for implementation
class CAXCtrl : public C0leControlModule
{
public:
```

NOTES

```
Cbitmap *m_CurrentBitmap, m_BitmapNight,m_BitmapDay;

// Constructor
public :
CAXCtrlCtrl ( );

// overrides
// ClassWizard generated virtual function overrides
// {{AFX_VIRTUAL ( ( CAXCtrlCtrl )
public:
virtual void Ondraw ( CDC* pdc, const CRect & rcBounds,
const CRect& rcInvalid );
virtual void DoPropExchange ( CPROPExchange* pPX );
virtual void OnResetState( );
// }}AFX_VIRTUAL
// Implementation
protected :
~ CAXCtrl( );
DECLARE_OLECREATE_EX(CAXCtrlCtrl )   // Class factory and
guid
DECLARE_OLETYPELIB ( CAXCtrlCtrl ) // Get TypeInfo
DECLARE_PROPPAGEIDS( CAXCtrlCtrl ) // Property page IDs
DECLARE_OLECTLTYPE ( CAXCtrlCtrl )   // Typename and misc
status

// Message maps
//{{AFX_MSG ( CAXCtrlCtrl )
afx_msg void OnLButtonUp ( UNIT nFlags, CPoint point );
// )} AFX_MSG
DECLARE_MESSAGE_map ( )

// Dispatch maps
//{{AFX_DISPATCH ( CAXCtrlCtrl )
BOOL m_show Frame;
Afx_msg void OnShowFrame Changed ( );
// }}AFX_DISPATCH
DECLARE_DISPATCH_MAP( )
Afx_msg void AboutBox ( );
// Event maps
//{{AFX_EVENT(CAXCtrlCtrl )
// }}AFX_EVENT_MAP( )
// Sispatch and event IDs
public :
enum{
```

NOTES

NOTES

```
//{{ AFX_DISP_ID(CAXCtrlCtrl )
dispidShowFrame=1L,
// }}AFX_DISP_ID
};
};
// {{ AFX_INSERT_LOCATION }}
// Microsoft Visual C++ will insert additional declarations
immediately before
// the previous line.
# endif
//                    !defined                    (
AFX_AXCTRLCTL_H_CC31D294_B1B1_11D1_80FC_00C0F6A83B7F_INCLUDED_
)

(4)

// AXCtrlCtl.cpp: Implementation of the CAXCtrlCtrl ActiveX
Control class.

# include "stdafx.h"
# include "AXCtrl.h"
# include "AXCtrlCtl.h"
# include "AXCtrlPpg.h"
# ifdef_DEBUG
# define new DEBUG_NEW
# undef THIS_FILE
static char THIS_FILE [ ] = _FILE_;
# endif

/////////////////////////////////////////////////////////
/////////////////////
// Message map
BEGIN_MESSAGE_MAP ( CAXCtrlCtrl, ColeControl )
//{{ AFX_MSG_MAP ( CAXCtrlCtrl )
ON_WM_LBUTTONUP ( )
// }} AFX_MSG_MAP
ON_OLEVERB (AFX_IDS_VERB_PROPERTIES, OnProperties )
END_MESSAGE_MAP ( )

/////////// //////// /////////// ////// /////////// ////
////// ////////// /////////////////
Dispatch map

BEGIN_DISPATCH_MAP(CAXCtrlCtrl, ColeControl)
/ /{{AFX_DISPATCH_MAP ( CAXCtrlCtrl )
```

```
DISP_PROPERTY_NOTIFY( CAXCtrlCtrl, 'ShowFrame",
m_showFrame, OnShowFrameChanged, VT_BOOL )
DISP_STOCKPROP_BACKCOLOR ( )
// }}AFX_DISPATCH_MAP
DISP_FUNCTION_ID ( CAXCtrlCtrl, " AboutBox",
DISPID_ABOUTBOX, AboutBox, VT_EMPTY,
VTS_NONE)
END_DISPATCH_MAP ( )

/////////// /////////// ///////// ////////// /////////
///////////
// Event map

BEGIN_EVENT_MAP ( CAXCtrlCtrl, ColeControl)
// {{AFX_EVENT_MAP ( CAXCtrlCtrl )
EVENT_STOCK_CLICK ( )
// }} AFX_EVENT_MAP
END_EVENT_MAP ( )

//////// /////////////// ///////// //////// /// //
// Property pages

// TODO : Add more property pages as needed. Remember to
increase the count!
BEGIN_PROPPAGEIDS (  CAXCtrlCtrl, 2 )
PROPPAGEID ( CAXCtrlPropPage : : guid )
PROPPAGEID ( CLSID_CcolorPropPage )
END_PROPPAGEIDS ( CAXCtrlCtrl )

/////////// ////// ////// //////////// ////////// /////
/// ///////
// Initialize Class factory and guid

IMPLEMENT_OLECREATE_EX         (CAXCtrlCtrl,
"AXCTRL.AXCtrlCtrl.1",
0xcc31d286,  0xb1b1, 0xlidl, 0x80, oxfc, 0, 0xc0, 0xf6,
0xa8, 0x3b, 0x7f )

/////////// /////////// ///////////// ////////// /////
///// ////// /////// ///////
// Type library ID and version

IMPLEMENT_OLETYPELIB (CAXCtrlCtrl, _tlid, _wVerMajor,
_wVerMinor )

/////// ////////// /////// //////// ////// // //////////
```

NOTES

NOTES

```
/ //// ////////
// Interface IDs
const IID BASED_CODE IID_DAXCtrl =
{ 0xcc31d284,  0xb1b1, 0xlidl, {0x80, oxfc, 0, 0xc0,
0xf6, 0xa8, 0x3b, 0x7f }};
const IID BASED_CODE IID_DAXCtrl Events=
{ 0xcc31d285,  0xb1b1, 0xlidl, { 0x80, oxfc, 0, 0xc0,
0xf6, 0xa8, 0x3b, 0x7f }};

////////////// //////// //////// ///////////////// /////
//////////// /////////////  ////////////
// Control type information
static const DWORD BASED_CODE  _dwAXCtrl ) leMisc =
OLEMIS_ACTIVATEWHENVISIBLE |
OLEMIS_SETCLIENTSITEFIRST |
OLEMIS_CANTLINKINSIDE |
OLEMIS_RECOMPOSEONRESIZE;
IMPLEMENT_OLECTLTYPE  (CAXCtrlCtrl,  IDS_AXCTRL,
_dWAXCtrlOleMISC )

//////////// //////////// /////////// //////////// /////
//////// //// ////////////
// CAXCtrlCtrl : : CAXCtrlCtrl Factory : : UpdateRegistry
-
//Adds or removes system registry entries for CAXCtrlCtrl

BOOL CAXCtrlCtrl : : CAXCtrlCtrl Factory : : Update Registry
( BOOL bRegister)
{
// TODO : Verify that your control follows apartment-
model threading rules.
// Refer to MFC TechNote 64 for more information.
/ / If your control does not conform to the apartment-
model rules , then
// you must modify the code below, changing the 6th
parameter from
// afxRegApartment Threading to 0.

If  ( bRegister )
Return Afx )leRegisterControlClass (
AfxGetInstanceHandle( ),
M_clsid,
m_lpszProgID,
IDS_AXCTRL,
IDB_AXCTRL,
AfxRegApartmentThreading,
```

NOTES

```
_dwAXCtrl0leMisc,
_tlid,
_wVerMajor,
_wVerMinor );
else
return Afx0leUnregisterClass ( m_clsid, m_lpszProgID );
}

//// ///////// /////////////// /////////////// //////
/ ///// ///////////
// CAXCtrlCtrl : : CAXCtrlCtrl – Constructor
CAXCtrlCtrl : : CAXCtrlCtrl ( )
{
InitializeIIDs ( &IID_DAXCtrl, &IID_DAXCtrlEvents );
//TODO: Initialize your control's instance data here.
m_BitmapNight.LoadBitmap ( IDB_BITMAP1 );
m_BitmapDay.LoadBitmap ( IDB_BITMAP2 );
m_CurrentBitmap = &m_BitmapNight; // initially display
the "night" bitmap
}

/////// ////// ///////////// ///////// ///////// ////
/ ///
// CAXCtrlCtrl : :  ~ CAXCtrlCtrl – Destructor
CAXCtrlCtrl : : CAXCtrlCtrl ( )
{
// TODO: Cleanup your control's instance data here.
}

/////// /////////// ///////// /////////// /////////
/ /////// ///////
// CAXCtrl : : OnDraw – Drawing function

void CAXCtrlCtrl : : OnDraw ( CDC* pdc, const Crest&
rcBounds, const CRect& rcInvalid)
{
// TODO : Replace the following code with your own drawing
code.
Cbrush Brush ( TranslateColor ( GetBackColor ( ) ) );
Pdc -> FillRect ( rcBounds, &Brush );

BITMAP BM;
CDC MemDC;

MemDC.CreateCompitableDC ( NULL);
```

NOTES

```
MemDC.SelectObject ( *m_CurrentBitmap);
M_CurrentBitmap -> GetObject ( size of ( BM ), &BM);
Pdc -> BitBlt
( ( rcBounds.right – BM.bmWidth ) /2,
( ( rcBounds.bottom – BM.bmHeight ) /2,
BM.bmWidth,
BM.bmHeight,
&MemDC,
0,
0,
SRCCOPY );

If ( m_showFrame )
{
Cbrush *p0leBrush = ( Cbrush * )pdc -> SelectStockObject
( NULL_BRUSH );
Cpen Pen (PS_SOLID | PS_INSIDEFRAME,  10, RGB ( 0,0,0));
Cpen  *poldPen = pdc - > SelectObject ( &Pen );
Pdc -> Rectangle ( rcBounds );
Pdc -> SelectObject ( poldPen );
Pdc -> SelectObject ( poldBrush );
}
}
///////// ////// ///////// /////////// //////////////
//// //// / /////////////////////
// CAXCtrlCtrl : : DoPropExchange – Persistance support
void CAXCtrlCtrl : :  DoPropExchange ( ( PropExchange *
pPX )
{
ExchangeVersion ( pPX, MAKELONG ( -wVerMinor, _wVerMajor
) );
ColeControl : : DoPropExchange (  pPX );

//TODO : Call PX_ functions for each Persistent custome
property.
PX_Bool ( pPX, T ( " ShowFrame" ), mshowFrame, FALSE);
}

///////////// //////////////// //////////////// ///
////////// /////////// //////////
// CAXCtrolCtrol : : OnResetState – Reset control to
default state
void CAXCtrol :: OnResetState ( )
{
ColeControl : : OnResetState ( ) ;
```

```
// Reset defaults found in DoPropExchange

// TODO : Reset any other control state here.
}

///////////// ///////// ////////// ////// /////// //
/////////// //////////
// CAXCtrlCtrl : : AboutBox - Display an "About" box to
the user
void CAXCtrlCtrl : : AboutBox ( )
{
CDialog dlgAbout (IDD_ABOUTBOX_AXCTRL);
DlgAbout.DoModal ( );
}

/////// /// //// ///// ///// //// //// /// /// /// /// /
// ///////////  //////////// /////////
//CAXCtrlCtrl message handlers
void CAXCtrlCtrl : : OnLButton Up (UNIT nFlags, CPoint
point )
{
// TODO : Add your message handler code here and / or call
default
if ( m_CurrentBitmap = = &m_BitmapNight)
   ( m_CurrentBitmap =  &m_BitmapDay)
else
( m_CurrentBitmap = = &m_BitmapNight)

InvalidateControl ( );

ColeControl : : OnLButtonUp (nFlags, point );
}
void CAXCtrlCtrl : : OnShowFrameChanged ( )
{
// TODO : Add notification handler code
InvalidateControl ( );
SetModifiedFlag ( );
}
_________ _______ _______ _______

(5)

// AXCtrlPpg.h : Declaration of the CAXCtrlPropPage
property page class.
#if                                          !defined
(AFX_AXCTRLPPG_H_CC31D296_B1B1_11D1_80FC_00C0F6A83B7F_INCLUDED_
```

NOTES

```
)
#defined  AFX_AXCTRLPPG_H_CC31D296_B1B1_11D1_80FC_
00C0F6A83B7F_INCLUDED_
```

NOTES

```
# if _MSC_VER>1000
# pragma once
# endif // MSC_VER>1000
///// /// ////// //// ///// //// ///// // //// // // ///
///// ////// //////// ///
// CAXCtrlPropPage : See AXCtrlPpg.cpp.cpp for
implementation.
class CAXCtrlPropPage : public C0lePropertyPage
{
DECLARE_DYNCREATE(CAXCtrlPropPage)
DECLARE_OLECREATE_EX(CAXCtrlPropPage)
// Constructor
public:
CAXCtrlPropPage ( );
// Dialog Data
//{{AFX_DATA (CAXCtrlPropPage)
enum { IDD=IDD_PROPPAGE_AXCTRL};
BOOL m_ShowFrame;
//}}AFX_DATA
// Implementation
Protected :
Virtual void DoDataExchange ( ( CdataExchange ( (
CdataExcahnge* pDX);
// DDX/DDV support
// Message maps
protected:
// {{AFX_MSG ( CAXCtrlPropPage)
// Note –ClassWizard will add and remove member
functions here.
// DO NOT EDIT what you see in these blocks of generated
code !
// }}AFX_MSG
DECLARE_MESSAGE_MAP ( )
};

//{{AFX_INSERT_LOCATION } }
//  Microsoft Visual C++ will insert additional declarations
immediately before
// the previous line.
#endif
//!defined ( AFX_AXCTRLPPG_H_CC31D296_B1B1_11D1_80FC_
```

```
00C0F6A83B7F_INCLUDED )
________ _______ _______ _________ ________ _______
(6)

AXCtrlPpg.cpp : Implementation of the CAXCtrlPropPage
property page class.
# include " stdafx.h"
# include "AXCtrl.h"
# include "AXCtrlPpg.h"
# ifdef -DEBUG
# define new DEBUG_NEW
# undef THIS_FILE
static char THIS_FILE [ ] = FILE ;
#endif

IMPLEMENT_DYNCREATE ( CAXCtrlPropPage, ColePropertyPage)

///// //////////// ///////////// //////////// ///////
//// //////
//Message map

BEGIN_MESSAGE_MAP ( CAXCtrlPropPage, ColePropertyPage)
// {{ AFX_MSG_MAP ( CAXCtrlPropPage)
// NOTE -ClassWizard will add and remove message map
entries
// DO NOT EDIT what you see in these blocks of generated
code !
// }} AFX_MSG_MAP
END_MESSAGE_MAP ( )
/////////////////// ///////////// ///////////// /////
//////// ////

// Initialize class factory and guid
IMPLEMENT_OLECREATE_EX    (CAXCtrlPropPage,
"AXCTRL.AXCtrlPropPage.1",
0xcc31d287,  0xb1b1, 0xlidl, 0x80, oxfc, 0, 0xc0, 0xf6,
0xa8, 0x3b, 0x7f )
/////// /////////// /////////// /////////// ///////
////////
// CAXCtrlPropPage : : CAXCtrlPropPage Factory : :
UpdateRegistry -
//Adds or removes system registry entries for
CAXCtrlPropPage

BOOL CAXCtrlPropPage : : CAXCtrlPropPage Factory : :
Update Registry ( BOOL bRegister)
```

NOTES

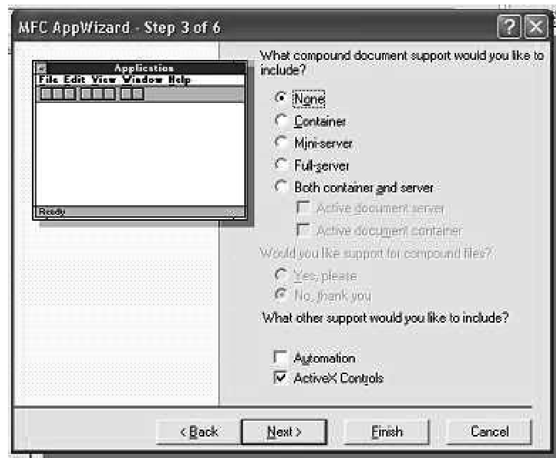
NOTES

```
{

If  (bRegister)
return AfxoleRegisterProperty PageClass (
AfxGetInstanceHandle( ),
m_clsid,
IDS_AXCTRL_PPG);
else
return Afx0leUnregisterClass (m_clsid, NULL);
}

/////////////////////////////////////////////////////
/////////////
// CAXCtrlPropPage : : CAXCtrlPropPage – Constructor
CAXCtrlPropPage : : CAXCtrlPropPage ( )
{
ColePropertyPage (IDD, IDS_AXCTRL_PPG_CAPTION)
{
//{{AFX_DATA_INIT(CAXCtrlPropPage)
m_ShowFrame = FALSE;
//}} AFX_DATA_INIT
}
////////////////// ///////////// //////////////// ////
////////// ///////////
CAXCtrlPropPage : : DoDataExchange – Moves data between
page and properties
Void CAXCtrlPropPage : : Do DataExchange ( CdataExchange
* pDX)
{
//{{AFX_DATA_MAP (( CAXCtrlPropPage)
DDP_Check  (pDX,  IDC_SHOWFRAME,  m_ShowFrame,  _T  (
“ShowFrame” ) );
DDX_Check(pDX, IDC_SHOWFRAME, m_ShowFrame);
//}}AFX_DATA_MAP
DDP_PostProcessing (pDX);
}
//// ///////////// ////////// //////////// /////////
/ //////
//CAXCtrlPropPage message handlers
```

4.6.1 Creating ActiveX Control Container Application

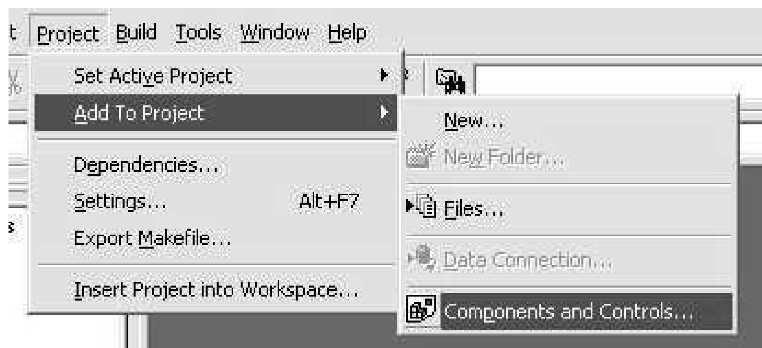
ActiveX container application is used to host the Automation server. The MFCC AppWizard is used to create an Automation controller application to host a calendar control.

In step 3 of AppWizard, the ActiveX Control option should be selected to use ActiveX control in the application.

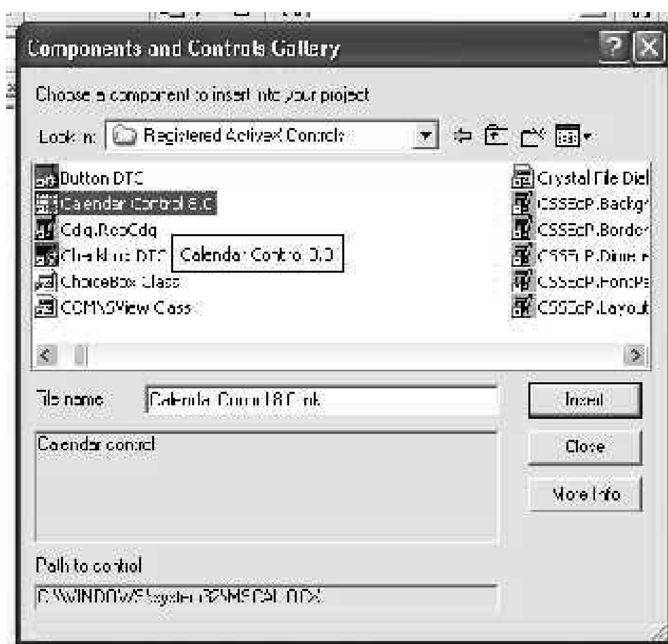


NOTES

Calendar control is already registered control of Microsoft that can be chosen to add to the project. Calendar control is installed in the project by selecting the Add to Project option from the Project menu and then selecting, components and controls.



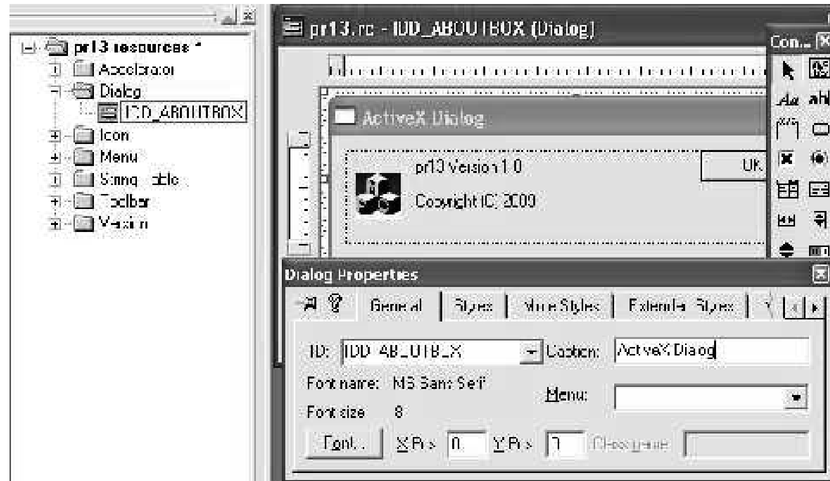
Select the already registered ActiveX control and then Calendar control.



ClassWizard generates two classes in the project directory CCalender and COleFont..

Add a dialog box to display the ActiveX control.

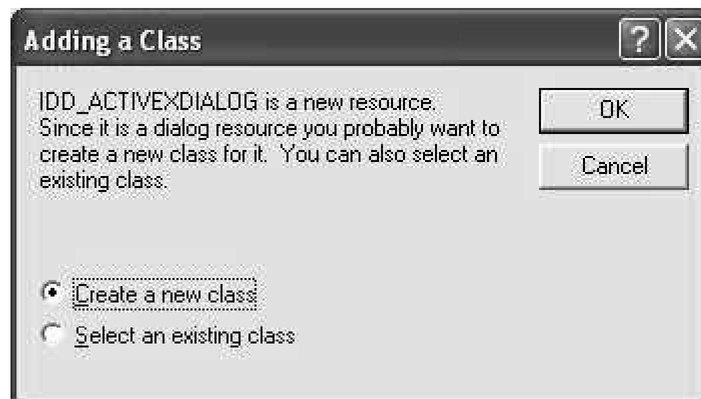
NOTES



Add controls so that the dialog box looks like as shown.

The Select Date button is used as the default button.

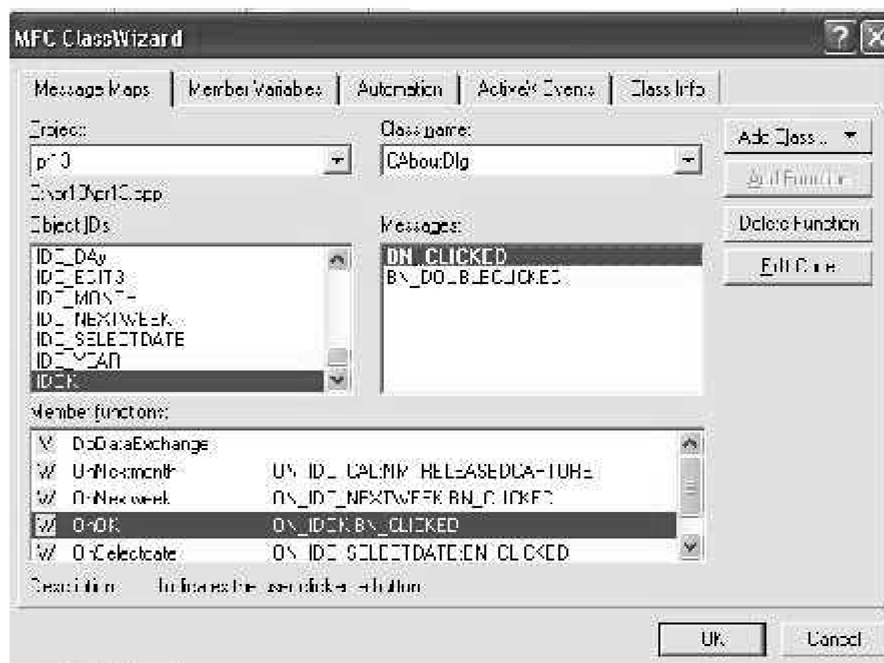
ClassWizard is used to create a class CActivedlg for the dialog box based on the template created.



Message handler functions are added to handle the events for the calendar control using the ClassWizard.

Object ID	Message	Message Handler Function
CAtivedlg	WM_INITDIALOG	ONInitDialog()
IDC_CAL	NextMonth	OnNextMonCal()
IDC_SELECTDATE	BN_CLICKED	OnSelDate()
IDC_NEXTWEEK	BN_CLICKED	OnNextWk()
IDOK	BN_CLICKED	OnOK()

NOTES



Member variables are added corresponding to the Calender, Day, Month and Year controls.

```
CAciveXDlg.h
#include "calendar.h"
// CActiveXDlg dialog
class CActiveXDlg : public CDialog

{
// Construction
public:
    CActiveXDlg(CWnd* pParent = NULL);   // standard
constructor

// Dialog Data
    //{{AFX_DATA(CActiveXDlg)
    enum { IDD = IDD_ACTIVEXDLG };
    CCalendar   m_calendar;
    short    m_sDay;
```

```
    short    m_sMonth;
    short    m_sYear;
    //}}AFX_DATA
```

NOTES

```
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CActiveXDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); //
DDX/DDV
                                                    //
  support
    //}}AFX_VIRTUAL

// Implementation
protected:

    //{afx_msg void OnReleasedcaptureCal(NMHDR* pNMHDR,
LRESULT* pResult);
    afx_msg void OnNextweek();
    afx_msg void OnSelectdate();
    virtual void OnOK();
    //}}AFX_MSG};
    DECARE_MESSAGE_MAP()

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional
//  declarations immediately before the previous line.

#endif  //  !defined(AFX_ACTIVEXDIALOG_H__
1917789D_6F24_11D0_ 8FD9_00C04FC2A0C2__INCLUDED_)

CActiveXDlg.cpp
//: implementation file
//

#include "stdafx.h"
#include "mymfc24.h"
#include "CActiveXDgl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[ ] = __FILE__;
#endif
```

```
/////////////////////////////////////////////////////
//////////////
// CActiveXDlg dialog

CActiveXDlg::CActiveXDlg(CWnd* pParent /*=NULL*/) :
CDialog(CActiveXDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CActiveXDlg)
    m_sDay = 0;
    m_sMonth = 0;
    m_sYear = 0;
    //}}AFX_DATA_INIT
}

void CActiveXDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CActiveXDlg)
    DDX_Control(pDX, IDC_CALENDAR1, m_calendar);
    DDX_Text(pDX, IDC_DAY, m_sDay);
    DDX_Text(pDX, IDC_MONTH, m_sMonth);
    DDX_Text(pDX, IDC_YEAR, m_sYear);
    //}}AFX_DATA_MAP
    DDX_OCColor(pDX, IDC_CALENDAR1, DISPID_BACKCOLOR,
m_BackColor);
}

BEGIN_MESSAGE_MAP(CActiveXDlg, CDialog)
    //{{AFX_MSG_MAP(CActiveXDlg)
    ON_BN_CLICKED(IDC_SELECTDATE, OnSelectDate)
    ON_BN_CLICKED(IDC_NEXTWEEK, OnNextWeek)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////
//////////////
// CActiveXDlg message handlers

BEGIN_EVENTSINK_MAP(CActiveXDlg, CDialog)
    //{{AFX_EVENTSINK_MAP(CActiveXDlg)
    ON_EVENT(CActiveXDlg, IDC_CALENDAR1, 3 /* NewMonth
*/, OnNewMonthCalendar1, VTS_NONE)
    //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()
```

NOTES

NOTES

```
BOOL CActiveXDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_calendar.SetValue(m_varValue); // no DDX for VARIANTs
    return TRUE;  // return TRUE unless you set the focus
to a control
                                // EXCEPTION: OCX Property
Pages should return FALSE
}

void CActiveXDlg::OnNewMonthCalendar1()
{
                              AfxMessageBox("EVENT:
CActiveXDlg::OnNewMonthCalendar1");
}

void CActiveXDlg::OnSelectDate()
{
    CDataExchange dx(this, TRUE);
    DDX_Text(&dx, IDC_DAY, m_sDay);
    DDX_Text(&dx, IDC_MONTH, m_sMonth);
    DDX_Text(&dx, IDC_YEAR, m_sYear);
    m_calendar.SetDay(m_sDay);
    m_calendar.SetMonth(m_sMonth);
    m_calendar.SetYear(m_sYear);
}

void CActiveXDlg::OnNextWeek()
{
    m_calendar.NextWeek();
}

void CActiveXDlg::OnOK()
{
    CDialog::OnOK();
    m_varValue = m_calendar.GetValue(); // no DDX for
VARIANTs
}
```

Initiate the dialog box containing the calendar control on any event of the main view window.

4.6.2 ActiveX Control Methods

An ActiveX control triggers various events to interact between itself and its associated control container. A container can likewise interact with a control via

NOTES

methods and properties. Methods are also known as functions. Methods and properties cater a transferred interface which is used for other applications to interact e.g., Automation clients and ActiveX control containers. Methods are similar to member functions of a C++ in terms of usage. There are two kinds of methods which can be implemented by the control. They are stock and custom. Same as the stock events, stock methods are the methods for which COleControl implementation can be provided. Custom methods are well-defined by the software developers and it permits supplementary tailoring of the control.

The Microsoft Foundation Class Library (MFC) introduced a method which permits required controls for providing appropriate support to stock and custom methods. The first portion is class COleControl. It is derived from CWnd. COleControl member functions are responsible for supporting stock methods which are very normal to every ActiveX controls. The second portion of this method is the dispatch map. A **dispatch map** is very much alike to a message map. There is a very small difference between the dispatch map and message map. It does not map a function to a Windows message ID. Dispatch map is used to map the virtual member functions to IDispatch IDS. To support numerous methods appropriately, a control class must be declared as a dispatch map. This is normally done with the help of the following code which resides in the header file of the control class.

```
DECLARE_DISPATCH_MAP()
```

The foremost requirement of the dispatch map is to initiate the association in between method names which is operated by an outside caller (e.g., container) and the member functions of the control's class which does the deployment of the methods. Post the dispatch map is accomplished, it requires to be classified in the control's deployment (CPP) file. The set of codes given below defines the dispatch map.

```
BEGIN_DISPATCH_MAP(CMyAxUICtrl, COleControl)
END_DISPATCH_MAP()
```

In case a project is created using MFC ActiveX Control Wizard, the above-mentioned codes are automatically appended. If we are not using any MFC ActiveX Control Wizard then the set of codes should be appended manually.

Returning Error Codes from a Method

It is possible to specify that an error has taken place within a method. To implement this, COleControl::ThrowError member function is used. It accepts an SCODE (status code) as a parameter. Predefined SCODE can be used. New SCODE can also be defined by the user.

ThrowError is only used at the time of returning an error from a property's Get or Set function. It can also generate error from an automation method. This is the instance when the suitable exception handler is existing on the stack.

```
The code for ThrowError is as follows:
void ThrowError(
    SCODE sc,
    UINT nDescriptionID,
    UINT nHelpID = -1);
```

```
void ThrowError(
     SCODE sc,
    LPCTSTR pszDescription = NULL,
     UINT nHelpID = 0);
```

NOTES

Where,

sc : The status code value to be conveyed. Some of the error codes are specified in the following table.

Error	Description
CTL_E_ILLEGALFUNCTIONCALL	Illegal function call
CTL_E_OVERFLOW	Overflow
CTL_E_OUTOFMEMORY	Out of memory
CTL_E_DIVISIONBYZERO	Division by zero
CTL_E_OUTOFSTRINGSPACE	Out of string space
CTL_E_OUTOFSTACKSPACE	Out of stack space
CTL_E_BADFILENAMEORNUMBER	Bad file name or number
CTL_E_FILENOTFOUND	File not found
CTL_E_BADFILEMODE	Bad file mode

nDescriptionID :The string resource ID of the exemption to be conveyed.

nHelpID :The help ID of the theme to be conveyed.

pszDescription :A string consisting of a clarification of the exemption to be conveyed.

ThrowError function should solitary be requested or called from a Get or Set function for an OLE property. It can also be called during the implementation of an OLE automation method.

If necessary, use the CUSTOM_CTL_SCODE macro to define a custom error code for a condition that is not covered by one of the standard codes. The parameter for this macro should be an integer between 1000 and 32767, inclusive. For example:

```
#define MYCTL_E_SPECIALERROR CUSTOM_CTL_SCODE(1000)
```

It may be required to deal with specific keystroke combinations in a typical manner. For example, pressing of Ctrl key and insert together or moving in between a set of edit controls during the usage of a directional key ID.

If the base class of the ActiveX control is COleControl, this can be supersed CWnd::PreTranslateMessage to deal with messages prior to the processing by the container. During this process, always TRUE has to be returned in case

addressing the message in the supersed of PreTranslateMessage. The following code is an example which portrays a feasible method of addressing any kind of messages which is in relation to the directional keys.

NOTES

```
BOOL CMyAxUICtrl::PreTranslateMessage(MSG* pMsg)
{
   BOOL bHandleNow = FALSE;

   switch (pMsg->message)
    {
   case WM_KEYDOWN:
      switch (pMsg->wParam)
       {
      case VK_UP:
      case VK_DOWN:
      case VK_LEFT:
      case VK_RIGHT:
         bHandleNow = TRUE;
         break;
       }
      if (bHandleNow)
       {
            OnKeyDown((UINT)pMsg->wParam, LOWORD(pMsg-
>lParam), HIWORD(pMsg->lParam));
       }
      break;
    }
   return bHandleNow;
}
```

4.6.3 ActiveX Events

ActiveX controls ideally uses events for alerting the container that an occurrence has taken place with the control. Example of such events can be the clicks made on the control, data inserted with the help of a keyboard, alterations in the situation of the control. Once any of the occurrence takes place, the control triggers an alert or event to make the container aware of. Events are also known as messages. There are two types of events or message supported by MFC which are as follows.

- Stock
- Custom

Stock events: The class COleControl deals this type of events automatically. Table 4.1 provides a comprehensive inventory of stock events.

NOTES

Table 4.1 *List of stock events*

Event	Firing function	Comments
Click	**void FireClick( )**	Triggered when the control captures the mouse, any **BUTTONUP** (left, middle, or right) message has arrived and the button is discharged over the control. The stock MouseDown and MouseUp events occur before this event. Event map entry: EVENT_STOCK_CLICK( )
DblClick	**void FireDblClick( )**	Same as Click but triggered when a **BUTTONDBLCLK** message has arrived. Event map entry: EVENT_STOCK_DBLCLICK( )
Error	**void FireError(SCODE** *scode* **, LPCSTR** lpszDescription **, UINT** nHelpID **= 0)**	Triggered when an inaccuracy takes place within your ActiveX control outside of the range of a method call or property access. Event map entry: EVENT_STOCK_ERROREVENT( )
KeyDown	**void FireKeyDown(short** nChar **, short** nShiftState **)**	Triggered when a WM_SYSKEYDOWN or WM_KEYDOWN message has arrived. Event map entry: EVENT_STOCK_KEYDOWN( )
KeyPress	**void FireKeyPress(short *** pnChar **)**	Triggered when a WM_CHAR message has arrived. Event map entry: EVENT_STOCK_KEYPRESS( )
KeyUp	**void FireKeyUp(short** nChar **, short** nShiftState **)**	Triggered when a WM_SYSKEYUP or WM_KEYUP message has arrived. Event map entry: EVENT_STOCK_KEYUP( )
MouseDown	**void FireMouseDown(short** nButton **, short** nShiftState **, float** *x* **, float** *y* **)**	Triggered in case any **BUTTONDOWN** (left, middle, or right) signal has arrived. The mouse signal is taken into control instantly prior to this event is triggered. Event map entry: EVENT_STOCK_MOUSEDOWN( )
MouseMove	**void FireMouseMove(short** nButton **, short** nShiftState **, float** *x* **, float** *y* **)**	Triggered when a WM_MOUSEMOVE message has arrived. Event map entry: EVENT_STOCK_MOUSEMOVE( )
MouseUp	**void FireMouseUp(short** nButton **, short** nShiftState **, float** *x* **, float** *y* **)**	Triggered in case any **BUTTONUP** (left, middle, or right) signal is received. The mouse signal is taken into control, is released before this event is triggered. Event map entry: EVENT_STOCK_MOUSEUP( )
ReadyStateChange	**void FireReadyStateChange( )**	Triggered when a control transitions to the subsequent set situation because of the volume of data received. Event map entry: EVENT_STOCK_READYSTATECHANGE( )

The custom events allow you to decouple the code that you want to execute after another piece of code completes. For example, you can separate the event listeners in a separate script. In addition, you can have multiple event listeners to

the same custom event. When a custom event is added, the Add Event Wizard does the required changes to the control class .H, .CPP, and .IDL files. The code given below is associated to the ClickIn event.

NOTES

The following code is appended to the header (.H) file of the respective control class:

```
void FireClickIn(OLE_XPOS_PIXELS xCoord, OLE_YPOS_PIXELS
yCoord)
{
  FireEvent(eventidClickIn, EVENT_PARAM(VTS_XPOS_PIXELS
VTS_YPOS_PIXELS), xCoord, yCoord);
}
```

This code illustrates an aligned function known as FireClickIn which calls COleControl::FireEvent with the ClickIn event and parameters which are declared using the Add Event Wizard. Moreover, the code mentioned below is appended to the event map for the required control. This is present in the implementation (.CPP) file of the control class.

```
EVENT_CUSTOM_ID("ClickIn", eventidClickIn, FireClickIn,
VTS_XPOS_PIXELS VTS_YPOS_PIXELS)
EVENT_CUSTOM_ID("ClickIn", eventidClickIn, FireClickIn,
VTS_XPOS_PIXELS VTS_YPOS_PIXELS)
```

This code maps the event ClickIn to the aligned function FireClickIn by passing the various parameters defined via the Add Event Wizard.

Finally, the code mentioned below is appended to the control's .IDL file.

```
[id(1)] void ClickIn(OLE_XPOS_PIXELS xCoord,
OLE_YPOS_PIXELS yCoord);
```

Triggered

The control class should map every event of the control to a specific member function which should be requested at the time when associated occurrence takes place to trigger events appropriately. This mapping methodology (known as **event map**) consolidates data about the event and consents Visual Studio to effortlessly approach and influence the events of the control. This event map is stated using the macro code mentioned below and can be found in the header (.H) file where the control class is declared.

```
DECLARE_EVENT_MAP()
```

It should be declared in the control's deployment (.CPP) file after the event map is provided. The codes mentioned below are used to define the event map which will permit the control to trigger precise events.

```
END_EVENT_MAP()
```

The following areas are included in ActiveX events:

- CommError
- Logon
- OnRemote
- Reposition
- Validate

NOTES

CommError

Applies to VAccess control

Description: (DEPRECATED - former I*net Data Server only.) This event triggers whenever an IDS communication error takes place. It will never trigger until the control is trying to interact with an IDS.

Syntax

```
Sub VAccess_CommError (ByVal bCanRecover As Boolean, ByVal
wsaeErrCode As Integer, ByVal errorString As String, bReTry
As Boolean)
```

Logon

Applies to VAccess control

Description: (DEPRECATED - former I*net Data Server only.) This event allows custom handling of the server logon related procedures. It will never be triggered till the control is logging into a secured IDS server and AutoLogon is declared as False.

Syntax

```
Sub VAccess_Logon(user As String, password As String,
database_set As String)
```

OnRemote

Applies to VAccess control

Description: (DEPRECATED - former I*net Data Server only.) This event is triggered at the time when the VAccess is incapable of connecting locally to a data file or data folder.

Syntax

```
Sub VAccess_OnRemote(byref goRemote As Boolean, byref
newLocation As String)
```

Reposition

Applies to VAccess control

Description: The Reposition event triggers subsequently a record operation alters the recent record which is pointed by the VAccess control. This event informs the program that the recent record related with the VAccess control has got modified. This event triggers after the new record gets converted as the up-to-date record.

Syntax

```
Sub VAccess_Reposition([Index As Integer])
```

Validate

Applies to VAccess control

Description: This event triggers prior to any record operation occurs on the related Zen file.

Syntax

```
Sub VAccess_Validate([Index As Integer,]OpCode As Integer,
InsertRecord As Integer, UpdateRecord As Integer)
```

NOTES

ActiveX Control Containers: Handling Events from an ActiveX Control

The following code initiates an event handler, named as OnClickInCircCtrl, for the Circ control's ClickIn event.

```
BEGIN_EVENTSINK_MAP(CContainerDlg, CDialog)
ON_EVENT(CContainerDlg, IDC_CIRCCTRL1, 1 /* ClickIn */,
OnClickInCircctrl1,
         VTS_I4 VTS_I4)
END_EVENTSINK_MAP()
```

Also, the following code is inserted to the CContainerDlg class implementation (.CPP) file for the event handler member function.

```
BOOL CContainerDlg::OnClickInCircctrl1(OLE_XPOS_PIXELS nX,
OLE_YPOS_PIXELS nY)
{
   // use nX and nY here
   TRACE(_T("nX = %d, nY = %d\n"), nX, nY);
   return TRUE;
}
```

Redistributing Visual C++ ActiveX Controls

Visual C++ provides ActiveX controls which can be used in applications and that can be distributed. During the distribution of applications, installation and registration of the .ocx for the ActiveX control (using Regsvr32.exe) is a mandatory. Moreover, it has to be ensured that the target system should have the latest versions of the below stated system files (* states that the file requires to be registered).

- Asycfilt.dll
- Comcat.dll *
- Oleaut32.dll *
- Olepro32.dll *
- Stdole2.tlb

Check Your Progress

12. How will you define the ActiveX control visual macros?
13. What is signed ActiveX controls?
14. What are the responsibilities of a COM server?
15. Define ActiveX control methods.

4.7 ANSWERS TO 'CHECK YOUR PROGRESS'

NOTES

1. An ActiveX control is a refillable software constituent which is founded on the "Component Object Model (COM)" which provides support to an extensive diversity of OLE functionality.
2. Browsers can be used to deactivate the ActiveX controls for various reasons. If ActiveX are filtered, browsers will restrict installing apps which uses ActiveX. This is certainly a perfect process for safe browsing.
3. Object Linking and Embedding (OLE) is technology from Microsoft. It is used to share various application data and objects formed in various formats from numerous sources. The term "Linking" creates a connectivity in between various object. The term "Embedding" performs the insertion or appending of data within an application.
4. The main advantages of OLE are as follows:
   - In case, changes are performed in source data, the same is available to the clients i.e., the updated data is always present to the client.
   - The primary application is not required to be in place for editing the associated data in the object. External editors can also be used.
   - All applications use the same interface to edit OLE data.
   - The user can choose his / her favourite editor and the same can be used to modify the object data
   - OLE is all about linking and embedding. Hence as no separate data is stored with the clients, disk space is saved.
5. Automation controllers are applications to access and manipulate the automation servers.
6. The Component Object Model (COM) is a standard communication protocol that allows objects to communicate through a special interface. COM technology provides the facility to access objects and services outside the application boundary.
7. The COM standard is language independent both at defining the COM object and defining the COM client, but there must be some official language for defining the interfaces and the COM class. COM uses an Interface Definition Language (IDL) for defining the interfaces.
8. The MIDL compiler has become a standard component of the Visual C++ environment. It is used to compile the source code into the C code which is then compiled into a project by the Visual C++ compiler.
9. A COM server is a component that resides in the EXE or DLL file and provides services to the COM client.
10. The four RPC libraries that must be included in the interface project are rpcndr.lib, rpcdce4.lib, rpcns4.lib and rpcrt4.lib.
11. The in-process COM object is faster because parameter marshalling is not required.

12. ActiveX controls are used extensively in the worksheet forms. It can be used with or without VBA code. Additionally, it is used on VBA UserForms. Normally, ActiveX controls are used to design flexible solutions which are catered by the Form controls. ActiveX controls consists of widespread attributes which is used to customize its appearance, associated behaviour, fonts and other features.
13. A signed ActiveX control specifies that the specific control is authentic and is certified by an established or identified authority and has never been modified since the authorization and authentication. Microsoft implemented the procedure of signing ActiveX controls for deploying specific volume of security to the components of the associated ActiveX controls.
14. Responsibilities of a COM server are:
    (i) A COM server is an object which caters service to an associated client. These services are always in the type of COM interface deployed and this can be requested by any client which is capable of receiving a pointer to the interfaces on the server object.
    (ii) A COM client is a code or object which is able to establish a pointer to a COM server. COM client uses the services provided by the COM server calling the procedures of its associated interfaces.
    (iii) COM servers are only association of compiled code which is called by executing processes.
    (iv) The server should instrument codes for a specific class object via installation of either the IClassFactory or IClassFactory2 interface.
15. An ActiveX control tiggers various events to interact between itself and its associated control container. A container can likewise interact with a control via methods and properties. Methods are also known as functions.

4.8 SUMMARY

- An ActiveX control is a refillable software constituent which is founded on the "Component Object Model (COM)" that provides support to an extensive diversity of OLE functionality.
- ActiveX container is a program which accepts ActiveX controls or document related objects.
- ActiveX document object is a document which is used to establish links or can be embedded within an ActiveX container.
- Browsers can be used to deactivate the ActiveX controls for various reasons. If ActiveX are filtered, browsers will restrict installing apps which uses ActiveX. This is certainly a perfect process for safe browsing.
- Object Linking and Embedding (OLE) is technology from Microsoft. It is used to share various application data and objects, formed in various formats from numerous sources. The term "Linking" creates a connectivity in between various object. The term "embedding" performs the insertion or appending of data within an application.

NOTES

- In case of linked object, data gets updated automatically once the source file is updated. The source file contains the data which is linked.
- In case, an Excel object is embedded data in the presentation will never change in case the source data is modifies.
- Automation servers are an ActiveX component that can be derived from other applications. They have one or more IDispatch Interface.
- An Automation controller is an application to access and manipulate the automation servers.
- Microsoft Word and Microsoft Excel are the examples of ActiveX document servers and the Internet Explorer is an example of an ActiveX document controller.
- COM is a standard communication protocol that allows objects to communicate through a special interface.
- COM object can be created in any language and can be used in a language-independent environment.
- The interface is the key which defines the object's behaviour irrespective of the operating system.
- A COM object must implement at least one interface, although the COM object can implement as many interfaces as it may require.
- The IUnknown interface is the base interface for all the other COM interfaces.
- COM uses an Interface Definition Language (IDL) for defining the interfaces.
- ActiveX control is a control which uses Microsoft ActiveX techniques.
- A signed ActiveX control specifies that the specific control is authentic and is certified by an established or identified authority and has never been modified since the authorization and authentication. Microsoft implemented the procedure of signing ActiveX controls for deploying specific volume of security to the components of the associated ActiveX controls.
- An ActiveX control tiggers various events to interact between itself and its associated control container. A container can likewise interact with a control via methods and properties. Methods are also known as functions.
- ActiveX controls ideally uses events for alerting the container that an occurrence has taken place with the control. Example of such events can be the clicks made on the control, data inserted with the help of a keyboard, alterations in the situation of the control. Once any of the occurrence takes place, the control triggers an alert or event to make the container aware of.

4.9 KEY TERMS

- **ActiveX Control:** An ActiveX control is a refillable software constituent which is founded on the "Component Object Model (COM)" which provides support to an extensive diversity of OLE functionality.

- **Object Linking and Embedding (OLE):** Object Linking and Embedding (OLE) is technology from Microsoft. It is used to share various application data and objects, formed in various formats from numerous sources. The term "Linking" creates a connectivity in between various object. The term "embedding" performs the insertion or appending of data within an application.
- **COM Object**: It is used to implement the related data and corresponding functions.
- **ActiveX Document:** It is an extension of object linking and embedding where the document has more control over the container application in which the document control is hosted.
- **Component Object Model:** It is a standard communication protocol that allows the objects to communicate through a special interface.
- **COM interfaces:** It consists of logically related well-defined methods that use the known parameters and return types.
- **IUnknown Interface:** It is the base interface for all the other COM interfaces.

4.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is the difference between an ActiveX control and COM object?
2. What are the basic components of an ActiveX control?
3. List the steps to use an ActiveX control in the application using AppWizard.
4. What is the purpose of GUIDGEN utility?
5. What provides COM objects to be reusable and interoperable?
6. What is the difference between message maps and interface maps?
7. How will you define the ActiveX control macros?
8. Define building an ActiveX server application.
9. When is an ActiveX control painted and repainted?
10. State the returning error codes from a method.
11. What do you mean by the ActiveX event?

Long-Answer Questions

1. Explain in detail the features of the ActiveX control.
2. What does an ActiveX technology mean? What are the types of ActiveX technologies?
3. What are the functionalities provided by the COM to support the component development?

NOTES

4. How are COM objects accessed? Explain.
5. Explain the nature and functions of an IUnknown interface?
6. What are the tools required to build a COM object? Explain their features.
7. Discuss briefly about the ActiveX control macros with the help of examples.
8. Elaborate on the building an ActiveX server application. Give appropriate examples.
9. Build a dialog-based ActiveX control to access the employee's names and designations of an organization and access it in a container application.
10. Analysis the returning error codes from a method with the help of examples.
11. Explain briefly about the ActiveX events. Give appropriate examples.

4.11 FURTHER READING

Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.

Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.

Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.

Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6*, 1st Edition. New Delhi: BPB Publications.

Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.

Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.

UNIT 5 INTERNET PROGRAMMING AND DATABASE APPLICATION

NOTES

Structure

5.0 Introduction
5.1 Objectives
5.2 Socket, MAPI and the Internet
  5.2.1 Creating a Socket Program
  5.2.2 Creating a Client Program
  5.2.3 Port
  5.2.4 Addresses
  5.2.5 Messaging Application Programming Interface (MAPI)
5.3 Internet Programming: Creating a Project
  5.3.1 Set_MERGE_PROXYSTUB
  5.3.2 The Build Rule - Understanding Custom Build Steps and Build Events
5.4 Active Template Library (ATL)
  5.4.1 Creation of the Project
5.5 Database Application
  5.5.1 ActiveX Data Objects (ADO)
  5.5.2 Database Application using ADO
5.6 Answer to 'Check Your Progress'
5.7 Summary
5.8 Key Terms
5.9 Self-Assessment Questions and Exercises
5.10 Further Reading

5.0 INTRODUCTION

A network socket is a software structure within a network node of a computer network that serves as an endpoint for sending and receiving data across the network. The structure and properties of a socket are defined by an Application Programming Interface (API) for the networking architecture. Sockets are created only during the lifetime of a process of an application running in the node. Because of the standardization of the TCP/IP protocols in the development of the Internet, the term network socket is most commonly used in the context of the Internet protocol suite, and is therefore often also referred to as Internet socket. In this context, a socket is externally identified to other hosts by its socket address, which is the triad of transport protocol, IP address, and port number. The term socket is also used for the software endpoint of node-Internal Inter-Process Communication (IPC), which often uses the same API as a network socket.

Messaging Application Programming Interface (MAPI) is an API for Microsoft Windows which allows programs to become email-aware. While MAPI is designed to be independent of the protocol, it is usually used to communicate with Microsoft Exchange Server.

An address is a collection of information, presented in a mostly fixed format, used to give the location of a building, apartment, or other structure or a plot of

land, generally using political boundaries and street names as references, along with other identifiers such as house or apartment numbers and organization name. Some addresses also contain special codes, such as a postal code, to make identification easier and aid in the routing of mail.

The Active Template Library (ATL) is a set of template-based C++ classes developed by Microsoft, intended to simplify the programming of Component Object Model (COM) objects. The COM support in Microsoft Visual C++ allows developers to create a variety of COM objects, OLE Automation servers, and ActiveX controls. ATL includes an object wizard that sets up primary structure of the objects quickly with a minimum of hand coding. On the COM client side ATL provides smart pointers that deal with COM reference counting. The library makes heavy use of the curiously recurring template pattern.

A database application is a computer software that retrieves data from a computerised database as its primary function. Information can be inserted, edited, or deleted from here, and it is then returned to the database. Accounting systems and airline reservation systems, such as SABRE, which were created starting in 1957, were early instances of database applications.

In this unit, you will learn about the socket, MAPI and the Internet, Internet programming, the active template library and database application.

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basic of socket, MAPI and the Internet
- Create a socket program
- Create a client social program
- Discuss about the internet programming
- Elaborate on the Active Template Library (ATL)
- Know about the various application of detabase

5.2 SOCKET, MAPI AND THE INTERNET

A socket is considered to be an endpoint of a bi-directional communication between two applications running on a network. A socket is associated to a specific port so that the TCP layer can recognize the program and send the data via the socket.

In C++, Socket programming is the method of linking two nodes (ideally SNMP enabled nodes) over a network so that the communication can take place at ease without any loss of data. During the creation of the socket, the program requires the socket type and the domain address and it is a mandatory. Usually, a server consists of a socket that is mapped to a particular port number. In this regard, the server tries to identify whether a client is trying to communicate or not and based on several conditions the server allows the client to establish connection or rejects the connections.

NOTES

The client is aware of the server's hostname and the appropriate port number which is used by the server to listen incoming requests from the client. In such case, there should also be an authentication mechanism by which the client introduces itself to the server. Once the authentication through, the client itself identifies the server port and maps it with a local port number which is used for the establishment of the connection. This is accomplished by the Network Operating System (NOS).

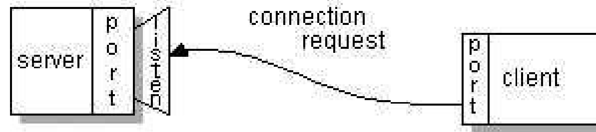


Fig. 5.1 *Network operating system*

One the acknowledgement takes place, the server accepts the client's connection request. As the server accepts the request, the server receives a new socket which is mapped with the same local port. Along with this the server is also aware of the client's socket and port number which will be used to establish the connection.

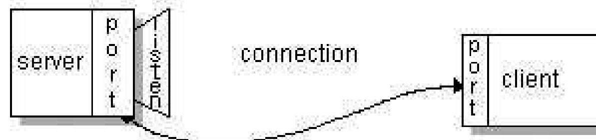


Fig. 5.2 *Client socket and port number*

On the client end, once the connection is established, a socket is effectively formed and the client uses this socket to communicate with the server. The client and server communicates using their respective assigned sockets.

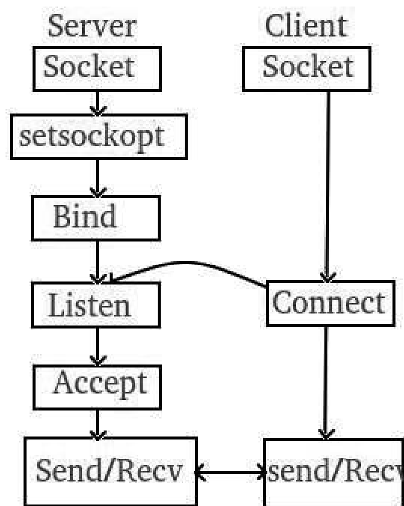


Fig. 5.3 *State diagram for Server and Client Model*

Techniques in Client-Server Communication

- **Socket**: Establishes a new communication.
- **Bind**: Attaches a local address to a specified socket.

NOTES

- **Listen**: Publicizes the preparedness to receive connections.
- **Accept**: Impede caller that specific time when a connection invitation reaches.
- **Connect**: Aggressively try to initiate a connection.
- **Send**: Send data by the established connection.
- **Receive**: Receive data by the established connection.
- **Close**: Established connection is released.

5.2.1 Creating a Socket Program

Int socketcr = socket (domain, type, protocol)

Socketcr = It is a sort of a form, an integer (file handle)

Domain = integer type, communication domain, example = AF_INET6 (IPv6 protocol)

Type = communication type

SOCK_DGRAM: UDP(unreliable, connectionless)

Protocol = Protocol number for Internet Protocol(IP), i.e. 0. This is the same number which is seen on the protocol field in the IP header of a packet.

Following code illustrates the server-side code which *echos* back the received message.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/time.h>

#define TRUE   1
#define PORT 5500

int main(int argc , char *argv[])
{
    int opt = TRUE;
   int master_sock , addrlen , new_sock , client_sock[30]
,
             maximum_clients = 30, act, i, value_read,
sock_descriptor;
   int maximum_socket_descriptor;
    struct sockaddr_in adr{};

    char buff[1025];  //data buffer of 1K
```

NOTES

```
    fd_set readfds; //set of socket file descriptors
    char *message = "ECHO Daemon v1.0 \\r\\n"; //message

    for (i = 0; i < maximum_clients; i++) //initialise
all client_sock to 0
    {
        client_sock[i] = 0;
    }
    if( (master_sock = socket(AF_INET , SOCK_STREAM , 0))
== 0) //creating a master socket
    {
        perror("Failed_Socket");
        exit(EXIT_FAILURE);
    }

    //These are the types of sockets that we have created
    adr.sin_family = AF_INET;
    adr.sin_addr.s_addr = INADDR_ANY;
    adr.sin_port = htons( PORT );

    if (bind(master_sock, (struct sockaddr *)&adr,
sizeof(adr))<0) //bind the socket to localhost port 5500
    {
        perror("Failed_Bind");
        exit(EXIT_FAILURE);
    }
    printf("Port having listener:  %d \\n", PORT);

    if (listen(master_sock, 3) < 0) //Specify 3 as maximum
pending connections for master socket
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    addrlen = sizeof(adr); //Accepting the Incoming
Connection
    puts("Looking For Connections");

    //*******************************//
    // Here we start using select functions and the macros
for multiple client handling

    while(TRUE)
    {
```

NOTES

```
        FD_ZERO(&readfds); //Clearing the socket set
       FD_SET(master_sock, &readfds); //Adding the master
socket to the set
        maximum_socket_descriptor = master_sock;

        for ( i = 0 ; i < maximum_clients ; i++) //Adding
child sockets to set
         {
          sock_descriptor = client_sock[i]; //Descriptor
for Socket

              if(sock_descriptor > 0) //if the socket
descriptor is valid then adding it to the read list
               FD_SET( sock_descriptor , &readfds);

                           if(sock_descriptor >
maximum_socket_descriptor) //Highest File Descriptor
Number which is needed for the select function
                         maximum_socket_descriptor =
sock_descriptor;
         }

        //Waiting for something to happen on the master
socket. As the wait time is NULL the wait is indefinite
         act = select( maximum_socket_descriptor + 1 ,
&readfds , nullptr , nullptr , nullptr);

        if ((act < 0) && (errno!=EINTR))
         {
            printf("Failed_Select");
         }
           if (FD_ISSET(master_sock, &readfds)) //Any
activity on the master socket is treated as an incoming
connection
         {
            if ((new_sock = accept(master_sock,
                                       (struct sockaddr
*)&adr, (socklen_t*)&addrlen))<0)
             {
                perror("Accept!");
                exit(EXIT_FAILURE);
             }

             //Informing the user of the socket number
which will be sued to send and receive messages
           printf("This is a New Connection, The socket
file descriptor is %d and the IP is : %s on Port : %d\\n"
```

```
                , new_sock , inet_ntoa(adr.sin_addr)
, ntohs
                 (adr.sin_port));

            if( send(new_sock, message, strlen(message),
0) != strlen(message)) // Sending Greeting Message on New
Connection
            {
                perror("Send!!");
            }
            puts("Welcome Text Sent Affirmative.");

             for (i = 0; i < maximum_clients; i++) //
Adding new socket to the array of sockets
            {
                if( client_sock[i] == 0 ) // Checking if
the position is empty
                {
                    client_sock[i] = new_sock;
                  printf("Adding new socket to the list
of sockets as %d\\n" , i);

                    break;
                }
            }
        }
        for (i = 0; i < maximum_clients; i++) //If not
the master socket then it is some i/o activity on some
other socket
        {
            sock_descriptor = client_sock[i];
            if (FD_ISSET( sock_descriptor , &readfds))
            {
             //Checking if the activity was for closing
and reading the incoming message
                if ((value_read = read( sock_descriptor
, buff, 1024)) == 0)
                {
                    //If someone disconnected, getting
their details and printing a message
                  getpeername(sock_descriptor , (struct
sockaddr*)&adr , \\
                        (socklen_t*)&addrlen);
                  printf("Disconnected Host. Their, IP
%s and PORT %d \\n",
                              inet_ntoa(adr.sin_addr) ,
ntohs(adr.sin_port));
```

NOTES

NOTES

```
                    close( sock_descriptor ); //Closing
the socket and marking it as 0 in the list to be reused
                    client_sock[i] = 0;
                }
              else //Echoing back the message that came
in the socket
                {
                    buff[value_read] = '\\0'; //Setting
the string terminating NULL byte on the end of the data
that is read
                         send(sock_descriptor , buff ,
strlen(buff) , 0 );
                }
            }
        }
    }
    return 0;
}
```

A socket gets opened on any specific TCP/IP 'Port'. Port is a unique number and is used as a communication endpoint. Port is of 16-bit and its value is unsigned. Explicitly, the number of TCP ports available are 65,535. Some ports are kept aside for specific purpose. Apart from these dedicated ports, others can be used for various purpose. Commonly used ports along with its associated services are listed below.

Port	Service
7	Ping
13	Time
15	Netstat
22	SSH
23	Telnet (default)
25	SMTP (Send mail)
43	Whois (Query information)
79	Finger (Query server information)
80	HTTP (Web pages)
110	POP (Receive mail)
119	NNTP
513	CLOGIN (Used for IP spoofing)

Socket Programming in C++

The following header file needs to be included in the program. This is used for enabling socket related features.

```
#include <sys/socket.h>
int socket ( int domain, int type, int protocol );
```

A Socket class is used to create a socket in programming in C++. Following is the process:

```
public Socket( InetAddress address,  int port )
throws IOException
```

Following is the list of functions related to Socket programming

- **Public InputStream getInputStream():** After the socket is created, it is essential to establish a method of receiving inputs from the users. This "Input Stream" function will return the "InputStream" which will enable to associate the data to this socket. This will also generate exceptions as required.
- **Public OutputStream getOutputStream():** After the socket is created, it is required to get an output from the user. This "output stream" function will return the "OutputStream" which will enable to associate the data to this socket.
- **Public synchronized void close():** It is essential to close it as it can't be kept opened. This function will close the socket.

Following are steps which is required during socket programming in C++.

- The socket needs to be created by delivering the appropriate domain, type and associated protocol.
- Setsockopted can be used in case it is required to reuse the address and port. It is not mandatory.
- On creation of the socket, "Bind" method should be used to map the socket to the address and the port number which is declared in the custom data structure.
- There is a method called listen which is normally used to ensure that the socket is inactive during the time it pauses for the client-server connectivity to get initiated.
- "Accept" method will obtain the very initial connection request. This is how the client and server get connected using the socket created for transferring data.

C++ Code for Socket Server

Following is the C++ code which illustrates socket programming from the server's side.

```
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/socket.h>
#include <std/lib.h>
#define PORT 8080
int main ( int argument, char const *argv[] )
{
```

NOTES

```
int obj_server, sock, reader;
struct sockaddr_in address;
int opted = 1;
int address_length = sizeof(address);
char buffer[1024] = {0};
char *message = "A message from server !";
if (( obj_server = socket ( AF_INET, SOCK_STREAM, 0)) ==
0)
{
pserror ( "Opening of Socket Failed !");
exit ( EXIT_FAILURE);
}
if ( setsockopted(obj_server, SOL_SOCKET, SO_REUSEADDR,
&opted, sizeof ( opted )))
{
pserror ( "Can't set the socket" );
exit ( EXIT_FAILURE );
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );
if (bind(obj_server, ( struct sockaddr * )&address,
sizeof(address))<0)
{
pserror ( "Binding of socket failed !" );
exit(EXIT_FAILURE);
}
if (listen ( obj_server, 3) < 0)
{
pserror ( "Can't listen from the server !");
exit(EXIT_FAILURE);
}
if ((sock = accept(obj_server, (struct sockaddr *)&address,
(socklen_t*)&address_length)) < 0)
{
pserror("Accept");
exit(EXIT_FAILURE);
}
reader = read(sock, buffer, 1024);
printf("%s\n", buffer);
send(sock , message, strlen(message) , 0 );
printf("Server : Message has been sent ! \n");
return 0;
}
```

5.2.2 Creating a Client Program

Following is the C++ code which illustrates socket programming from the client end.

NOTES

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080
int main ( int argument, char const *argv[] )
{
int obj_socket = 0, reader;
struct sockaddr_in serv_addr;
char *message = "A message from Client !";
char buffer[1024] = {0};
if (( obj_socket = socket (AF_INET, SOCK_STREAM, 0 )) <
0)
{
printf ( "Socket creation error !" );
return -1;
}
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
// Converting IPv4 and IPv6 addresses from text to binary
form
if(inet_pton ( AF_INET, "127.0.0.1",
&serv_addr.sin_addr)<=0)
{
printf ( "\nInvalid address ! This IP Address is not
supported !\n" );
return -1;
}
if ( connect( obj_socket, (struct sockaddr *)&serv_addr,
sizeof(serv_addr )) < 0)
{
Printf ("Connection Failed: Can't establish a connection
over this socket !" );
return -1;
}
send ( obj_socket , message , strlen(message) , 0 );
printf ( "\nClient : Message has been sent !\n" );
reader = read ( obj_socket, buffer, 1024 );
printf ( "%s\n",buffer );
return 0;
}
```

Socket programming can also be accomplished using Python, Java and PHP as well.

NOTES

Server/Client Applications

The fundamental steps for client-server setup are as follows:

- A client application sends a service request to a server application.
- The server application sends acknowledgement.
- The connection is established.
- Some of the basic data communications between client and server are as follows:
  - File transfer – This activity is used to receive file/files based on request.
  - Web page - requests the URL and receives an associated web page.
  - Echo – Client sends a message and receives it back.

Server Socket

- Creation of a socket - File descriptor to be received.
- Binding the socket to a specific address -need to specify the port which is to be used for communication.
- Listening on a port and pause for a connection to be taking place.
- Acceptance of the requested connection from the client.
- send/receive – Sending and receiving of data i.e., the communication between client and server.
- Shutdown or closure to the communication.
- Releasing of data is stopped.

Client Socket

- Creation of a socket
- Connection established with a server.
- Sending /receiving – transmission of data
- Shutdown or closure to the communication
- Releasing of data is stopped.

5.2.3 Port

A port is a communication endpoint. At the software level, a port is a logical construct that identifies a specific process or a type of network service. A port is identified for each transport protocol and address combination by a 16-bit unsigned number, known as the port number. The most common transport protocols that use port numbers are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

A port number is always associated with an IP address of a host and the type of transport protocol used for communication. It completes the destination or origination network address of a message. Specific port numbers are reserved to identify specific services so that an arriving packet can be easily forwarded to a running application. For this purpose, port numbers lower than 1024 identify the historically most commonly used services and are called the well-known port numbers. Higher-numbered ports are available for general use by applications and are known as ephemeral ports.

Ports provide a multiplexing service for multiple services or multiple communication sessions at one network address. In the client server model of application architecture, multiple simultaneous communication sessions may be initiated for the same service.

5.2.4 Addresses

A socket is a combination of ports and IP addresses. An Internet Protocol address (IP address) is the logical address of our network hardware by which other devices identify it in a network. An Internet Protocol (IP) is a unique address which provides a universal address across the network. It is addressed to the data packets which transmit over the network working with the IP protocol. The IP address consists of four parts and each is separated by a dot. An IP address is assigned to every device that is connected to internet for its unique identification.

Features of IP address

- It provides a unique address over a network. We cannot get the same IP address for two system units. In case an IP address is set as same for two systems, an IP conflict process takes place wherein the data packets do not know the destination place.
- An IP address contains a default network whose address is 0.0.0.0 which is used to the default network.

5.2.5 Messaging Application Programming Interface (MAPI)

MAPI stands for Messaging Application Program Interface. The name itself we can understand that, this has something to do with mail. MAPI is a Microsoft Windows program interface which is used to send mails across windows application and documents can be attached within the mails.

MAPI consists of applications like word processors, spreadsheets, presentation and graphics applications. All MAPI-compatible applications consist of a Send Mail which is used to send mails. It comprises of a set of C/C++ language functions which are as usual warehoused in a program library called as a Dynamic Link Library (DLL). Developers refers to the MAPI library for using Microsoft's Collaboration Data Objects (CDO) during coding of Microsoft's Active Server Page (ASP). The CDO library is available with Microsoft's Internet Information Server (IIS). Eudora is an e-mail application which consists of a MAPI server.

NOTES

Features of MAPI

- MAPI is a protocol for client only.
- It is used to access mailbox via Outlook or MAPI enabled email clients.
- By default, MAPI is enabled in email clients.
- In case MAPI is disabled, users will not be able to access their mailbox.
- MAPI doesn't restrict users from using Outlook using other protocols (e.g., POP3, IMAP4, etc.) for accessing their mailbox.

Benefits of MAPI over HTTP

MAPI over HTTP have the following advantages to the clients.

- Enables authentication via HTTP protocol.
- Delivers quicker reconnection times after there is break in the communications. Examples of a communication break include:
  - o In case the device has gone in hibernation mode.
  - o Switching from a wired network to a wireless network.
- Provides a session context which is not at all reliant on the established connection.

MAPI Programming Overview

MAPI references rewritten for C and C++ developers with a variation requirements and knowledge with messaging. MAPI is a widespread collection of functions that is used by the developers during developing mail-enabled applications.

Example of MAPI program

The following MAPI program consists of a function which permits coders to access a user's mail messages via programs. The following code is developed using Microsoft Visual C++.

NOTE: The following code implements MAPI.DLL as a static library.

```
/* readmail.c */
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <mapi.h>
#include <string.h>

int readmail();

long err;
LHANDLE lhSession;
lpMapiMessage FAR *lppMessage;
lpMapiMessage lpMessage;
char szSeedMessageID[512];
char szMessageID[512];
```

NOTES

```
char szTmp[4096];
char szTmp2[50];

LPSTR lpszSeedMessageID=&szSeedMessageID[0];
LPSTR lpszMessageID=&szMessageID[0];

int main()
{
 readmail();
 return(0);
}

int readmail()
{

  /************ Logon *********************/
  err = MAPILogon(0L, "", "", MAPI_LOGON_UI, 0L,
  &lhSession);
  if(err != SUCCESS_SUCCESS)
   {
    MessageBox(0, "Error logging on", "Error", MB_OK);
     return(0);
   }

 /********* Find Messages *******************/
 *lpszSeedMessageID = '\0';

  // reset MAPIFindNext back to the top again
   err = MAPIFindNext(lhSession, 0L, "IPM.Microsoft
Mail.Note",
  lpszSeedMessageID, 0L, 0L, lpszMessageID);
     if(err != SUCCESS_SUCCESS)
     {
        MessageBox(0, "Error finding first message",
"Error",
       MB_OK);
       err = MAPILogoff(lhSession, 0L, 0L, 0L);
       return(0);

     }

   do
   {
    lppMessage=(lpMapiMessage FAR *) &lpMessage;
```

NOTES

```
/******** Read Message *************/
err = MAPIReadMail(lhSession, 0L, lpszMessageID,
MAPI_PEEK, 0L, lppMessage);
if(err != SUCCESS_SUCCESS)
 {
     MessageBox(0, "Error during message read",
"Error",
  MB_OK);
  err = MAPILogoff(lhSession, 0L, 0L, 0L);
  return(0);
 }
  /******* copy Subject and NoteText into string
*************/
  if((lpMessage->lpszSubject) != NULL){
   _fstrcpy(szTmp2, lpMessage->lpszSubject);    /
/  Check for NULL
   strings
   }
  else _fstrcpy(szTmp2,"No subject text");
  if((lpMessage->lpszNoteText) != NULL){
   if(_fstrlen(lpMessage->lpszNoteText)>4096){   /
/ Check for large
   message body
         MessageBox(0, "Message body to large",
"MAPI2.C", MB_OK);
     }
   _fstrcpy(szTmp, lpMessage->lpszNoteText);
   }
  else _fstrcpy(szTmp, "No message body");
  printf("\nSUBJECT: %s\n", szTmp2);
  printf("\nNOTETEXT: \n%s\n", szTmp);
         /****** free memory used by MAPI
********************/
  err = MAPIFreeBuffer(lpMessage);
   if(err != SUCCESS_SUCCESS)
   {
         MessageBox(0, "Error freeing memory",
"Error",MB_OK);
   }
 //get next message ID.
 lstrcpy(lpszSeedMessageID, lpszMessageID);
 err = MAPIFindNext(lhSession, 0L, "IPM.Microsoft
Mail.Note",
  lpszSeedMessageID, 0L, 0L, lpszMessageID);
 if(err != SUCCESS_SUCCESS)
```

```
      {
       MessageBox(0, "No more messages", "Warning",
        MB_OK);
       err = MAPILogoff(lhSession, 0L, 0L, 0L);
        return(0);
      }
     lppMessage=(lpMapiMessage FAR *) &lpMessage;

  }while(err == SUCCESS_SUCCESS);

 /************** Logoff ***************/
  err = MAPILogoff(lhSession, 0L, 0L, 0L);
  return(0);
```

NOTES

Difference between MAPI and SMTP

- SMTP is used to send mails only. MAPI is used to send as well as receive mails.
- SMTP is more extensive than MAPI.
- SMTP is completely autonomous of clients but MAPI is not.
- MAPI is responsible to save a copy of mail sent whereas SMTP do not save it.

The new API functions for MAPI are as follows:

Logon/Logoff

- MapiLogoff: Ends a MAPI session which was initiated with MapiLogon.
- MapiLogon: Creates a new MAPI Session. Logon credentials i.e., MAPI profile name and a password to logon to a new MAPI session has to be provided in this case.

Sending

- MapiSendMessage: Creates a new mail/message in the active session and forwards it immediately.
- MapiSendMessageEx: Forwards a mail/message which was created with MapiCreateMessage.

Creating/Deleting/Saving

- MapiCreateMessage: A new message is created and it returns a handle for that specific mail/message.
- MapiDeleteMessage: Used to remove or delete a mail.
- MapiSaveMessage: Saves alteration to a mail/message.

NOTES

Accessing Message Properties

- MapiGetLastError: It returns the error code of the last MAPI call.
- MapiGetMessageAttachment: Extracts details about an attachment within a mail/message.
- MapiGetMessageProperty: Extracts message property from a MAPI mail/message.
- MapiGetMessageRecipient: Extracts information about the recipient of a message.
- MapiGetMessage: From an active MAPI Session, this function extracts a message via the Message ID.
- MapiGetNextMsgId: This function returns the subsequent mail/Message ID in the Inbox of the associated MAPI Session.
- MapiSetMessageAttachment: Establishes information with regards to a mail/message attachment.
- MapiSetMessageProperty: Establishes a message property value of a MAPI mail/message.
- MapiSetMessageRecipient: Establishes information about the recipient of a mail/message.

Miscellaneous Functions

- MapiInit: Initializes MAPI Support for the existing virtual users.
- MapiExit: Unloads MAPI support for existing virtual users.
- MapiFreeMessage: Releases memory which is tied to the message handle of the virtual user.

Another Example of MAPI

The following program is written in the new MAPI API and is used to perform the following activities:

- Login to a MAPI session
- Reading of every mail within the Inbox
- Removal of the first mail from the Inbox
- Modifying the subject of the first mail
- Sending of a new mail

```
benchmark BenchmarkName
use "kernel.bdh"
use "mapi.bdh"

var
  ghSession : number init 0;
// Workload Section
dcluser
  user
```

NOTES

```
    VirtUser
  transactions
    TInit          : begin;
    TGetMsgs       : 1;
    TSendMsg       : 1;
    TSendMsgEx     : 1;
    TDeleteMsg     : 1;
    TChangeMsg     : 1;
    TEnd           : end;

// Transactions Section
dcltrans
  transaction TInit
  begin
    MapiInit();
    ghSession := MapiLogon("Outlook", null);
    if(ghSession = 0) then halt; end;
  end TInit;

  transaction TGetMsgs
  var
    hMsg, nRecipCount, nAttachCount, i : number;
    sLastMsgId, sNextMsgId             : string;
    sValue                             : string;
  begin
    sLastMsgId := "";
      // Iterate through all Mails and retrieve mail
properties
       while(MapiGetNextMsgId(ghSession, sLastMsgId,
sNextMsgId)) do
     hMsg := MapiGetMessage(ghSession, true, false, false,
false, sNextMsgId);

     MapiGetMessageProperty(hMsg, MAPI_PROP_MSG_MSGTYPE,
sValue);
      writeln("MsgType: " + sValue);
     MapiGetMessageProperty(hMsg, MAPI_PROP_MSG_SUBJECT,
sValue);
      writeln("Subject: " + sValue);
                   MapiGetMessageProperty(hMsg,
MAPI_PROP_MSG_DATERECEIVED, sValue);
      writeln("Date: " + sValue);
                   MapiGetMessageProperty(hMsg,
MAPI_PROP_MSG_SENDER_NAME, sValue);
      writeln("Send-Name: " + sValue);
```

```
                    MapiGetMessageProperty(hMsg,
MAPI_PROP_MSG_SENDER_ADDRESS, sValue);
     writeln("Sender-Address: " + sValue);

     // Iterate through all Recipients
              if(MapiGetMessageProperty(hMsg,
MAPI_PROP_MSG_RECIPIENT_COUNT, sValue)) then
       nRecipCount := number(sValue);
       writeln("RecipientCount: " + sValue);

       for i:=0 to nRecipCount-1 do
       MapiGetMessageRecipient(hMsg, i, true, sValue);
        writeln("Recipient-Name: " + sValue);
       MapiGetMessageRecipient(hMsg, i, false, sValue);
        writeln("Recipient-Address: " + sValue);
       end;
     end;

     // Iterate through all Attachments
              if(MapiGetMessageProperty(hMsg,
MAPI_PROP_MSG_ATTACH_COUNT, sValue)) then
       nAttachCount := number(sValue);
       writeln("AttachCount: " + sValue);

       for i:=0 to nAttachCount-1 do
       MapiGetMessageAttachment(hMsg, i, true, sValue);
        writeln("Attach-File: " + sValue);
             MapiGetMessageAttachment(hMsg, i, false,
sValue);
        writeln("Attach-Path: " + sValue);
       end;
     end;

    writeln("——————————————————————————");

     // Free the message
     MapiFreeMessage(hMsg);
     sLastMsgId := sNextMsgId;
   end;
 end TGetMsgs;

 transaction TSendMsg
 begin
   // Send a new message
   MapiSendMessage(
```

NOTES

NOTES

```
      ghSession,
      "TestSubject",
      "Test Message Text",
      "",
      "SenderName",
      "sender@sendersdomain",
      "RecipientName",
      "recipient@recipientdomain");
  end TSendMsg;

  transaction TSendMsgEx
  var
    hMsg : number;
  begin
    // Create a new message
    hMsg := MapiCreateMessage(
      "TestSubject",
      "Test Message Text",
      "",
      "SenderName",
      "sender@sendersdomain",
      "RecipientName",
      "recipient@recipientdomain");
    if(hMsg = 0) then halt; end;

    // Add a 2nd Recipient
                MapiSetMessageProperty(hMsg,
MAPI_PROP_MSG_RECIPIENT_COUNT, "2");
    MapiSetMessageRecipient(hMsg, 1, "2nd Recipient",
"test@test.com", MAPI_MSG_RECIPTYPE_CC);

    // Add an attachment
                MapiSetMessageProperty(hMsg,
MAPI_PROP_MSG_ATTACH_COUNT, "1");
      MapiSetMessageAttachment(hMsg, 0, "test.txt",
"c:\\temp\\test.txt");

    // Send the message
    MapiSendMessageEx(ghSession, hMsg);

    // Free the message
    MapiFreeMessage(hMsg);
  end TSendMsgEx;

  transaction TDeleteMsg
```

NOTES

```
 var
   sMsgId : string;
   hMsg   : number;
 begin
   // delete the first message
   if MapiGetNextMsgId(ghSession, "", sMsgId) then
     MapiDeleteMessage(ghSession, sMsgId);
   end;
 end TDeleteMsg;

 transaction TChangeMsg
 var
   sMsgId, sSubject : string;
   hMsg   : number;
 begin
   // retrieve the first message and change the message
subject
   if MapiGetNextMsgId(ghSession, "", sMsgId) then
     hMsg := MapiGetMessage(ghSession, true, false, true,
true, sMsgId);
      if(hMsg <> 0) then
       // get the subject - add the value 'Modified' and
set the new value
                    MapiGetMessageProperty(hMsg,
MAPI_PROP_MSG_SUBJECT, sSubject);
        sSubject := sSubject + " Modified";
                    MapiSetMessageProperty(hMsg,
MAPI_PROP_MSG_SUBJECT, sSubject);

        // Save the changes
       MapiSaveMessage(ghSession, hMsg, sMsgId);

        // Free Message
       MapiFreeMessage(hMsg);
      end;
   end;
 end TChangeMsg;

 transaction TEnd
 begin
   // Logoff
   MapiLogoff(ghSession);
   MapiExit();
 end TEnd;
```

Check Your Progress

1. Define the term socket.
2. What are the technique in client-serve communication?
3. What are the fundamental steps for client-server setup?
4. What is port?
5. Define the term addresses.
6. What do you understand by the MAPI?

NOTES

5.3 INTERNET PROGRAMMING: CREATING A PROJECT

Following are the steps to create a project and add a source file.

Creating a C++ project in Visual Studio

a) From the "main menu", choose "File'! New '! Project" for opening the "Create a New Project" dialog box.

b) At the upper portion of the dialog, "set Language to C++", "set Platform to Windows", and set "Project type to Console".

c) From the "filtered list" of the "project types", "Console App" should be chosen and then "Next" button should be pressed. In the next page, a name for the project should be entered and the project location should be specified in case it is required.

d) The "Create" button should be chosen for creating the project.

Adding a new source file

a) Invoke "Solution Explorer".

b) The process of adding a new source file to the project is as stated below.
   I. Right-click the "Source Files" folder in "Solution Explorer", point to "Add", and then click on "New Item".
   II. In the "Code node", the C++ File (.cpp) should be clicked, a name for the file should be provided and then the "Add" button should be clicked.

c) The .cpp file displays in the "Source Files" folder in "Solution Explorer" and the file is opened in the "Visual Studio" editor.

d) In the "file" within the "editor", a valid C++ program should be provided which uses the "C++ Standard Library", or a sample program code can be copied in the file.

e) The file should now be saved.

f) Click on the "Build Solution" on the "Build" menu.

g) The "Output window" portrays the information related to the progress of the compilation of the program.

h) Click on "Start" without "Debugging" on the "Debug" menu.

NOTES

Creating a Visual C++ Source Code and Compiling it on the Command Line

In the "developer command prompt" window, create a folder as follows:

```
md c:\hello — To create a folder
cd c:\hello — To change to that folder.
```

a) Enter the notepad "hello.cpp" in the command prompt window.

b) "Yes" should be chosen at the time notepad prompts for creating a new file. This process will open an empty notepad window, which is prepared for entering the program code in a file "hello.cpp".

c) In the notepad, following code should be entered:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, world, from Visual C++!" << endl;
}
```

This program will write one line of text on the monitor and then exit.

d) "Save" the work in the "Notepad"" on the "File" menu, choose "Save".

"hello.cpp" is now ready for compilation.

e) Now, the "developer command prompt" window should be accessed. *dir* command should be entered at the command prompt to find the content of the c:\hello folder. The source file hello.cpp exists in the folder, which looks as follows:

```
c:\hello>dir
 Volume in drive C has no label.
 Volume Serial Number is CC62-6545

Directory of c:\hello

05/24/2016 05:36 PM    <DIR>          .
05/24/2016 05:36 PM    <DIR>          .
05/24/2016 05:37 PM              115 hello.cpp
              1 File(s)            115 bytes
             2 Dir(s) 571,343,446,016 bytes free
```

f) On the "developer command prompt", "cl /EHsc hello.cpp" should be entered for compiling the program.

"cl.exe" is a compiler which produces an .obj file which contains the compiled code and then executes the linker to form an executable program named "hello.exe".

This name will be seen in the lines of output information which the compiler puts it on the monitor. The output of the compiler should be as follows:

```
c:\hello>cl /EHsc hello.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.10.25017
for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

hello.cpp
Microsoft (R) Incremental Linker Version 14.10.25017.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:hello.exe
hello.obj
```

g) To execute the hello.exe program, from the command prompt, enter “hello”.

The program will display the text mentioned below and exit:

Hello, world, from Visual C++!

To compile the program which contains supplementary source code files, the following command should be used.

cl /EHsc file1.cpp file2.cpp file3.cpp

5.3.1 Set_MERGE_PROXYSTUB

Selection of a COM-based DLL from the ATL COM AppWizard results the following files for creating a COM-based DLL.

- <project name>.cpp consists of DllMain entry point along with the Dllxxx functions COM considers. This file gratifies the calls to the normal functions via deputation to the CComModule class which is a portion of the project.
- <project name>.def specifically exports DllGetClassObject, DllCanUnloadNow, DllRegisterServer, and DllUnregisterServer. These are the hooks which COM considers to be identified within the DLL.
- <project name>.idl specifies an empty IDL file which is waiting for certain interface characterizations.
- <project name>.rc consists of resources which is required to be included within the COM-based DLL.
- stdafx.h is the header file where normal #include statements are provided.
- stdafx.cpp is responsible for compiling the stdafx.h file.
- resource.h consists of identifiers which is required by the resource script.
- <project name>.clw is normally used by ClassWizard.

Following three files are created during compiling of the project.

- <project name>.h is a header file which consists of the C++ versions of the COM interfaces originating in <project name>.idl.

- <project name>_i.c consists of the GUIDs which are specified in <project name>.idl.
- <project name>.tlb is the binary explanation of the COM-based server.

NOTES

Along with the source code files listed on top, project files for a proxy/stub DLL are also received and they are based on the contents of <project name>.idl. The proxy/stub source code comprises of three additional files which are as follows:

- <project name>ps.def
- <project name>ps.mk
- dlldata.c (This file is generated in case <project name>.idl has interfaces defined within it.)

With respect to clients, an ATL-based COM DLL is a normal COM DLL. The differentiation is that various implementation details are addressed internally by ATL.

5.3.2 The Build Rule - Understanding Custom Build Steps and Build Events

With respect to Visual C++ development environment, there are three primary methods to customize the build process. They are as follows:

Custom Build Steps: This is a build rule which is related with a project. A "Custom Build Step" can mention command lines for execution, any supplementary input or output files and a message for displaying.

Custom Build Tools: It is a build rule which is related to one or more files. A "custom build" step can distribute input files to a "custom build tool" and the consequence is one or more output files. For example, the help files in an MFC are developed using custom build tool.

Build Events: "Build events" permits to customize a project's build. There are three build events: "**pre-build"**, "**pre-link"** and "**post-build"**. A build event enables to declare an action to happen at a given point of time in the build process. For example, "build event" can be used to register a file with regsvr32.exe post the building of the project is accomplished.

For every project within a solution, "build events" and "custom build steps" gets executed in the following sequence with supplementary "build steps".

a) Pre-Build event
b) Custom build tools on individual files
c) MIDL
d) Resource compiler
e) The C/C++ compiler
f) Pre-Link event
g) Linker or Librarian (as appropriate)
h) Manifest Tool
i) BSCMake

j) Custom build step on the project

k) Post-Build event

The "custom build step" on the project and a "post-build event" gets executed successively after all other build processes gets finished.

NOTES

Formatting the Output of a Custom Build Step or Build Event

- Warnings and errors are displayed in the Output window.
- Output appears in the "Task List" window.
- The appropriate topic is displayed by clicking on the output in the "Output window".
- F1 processes are permitted in the "Task List" window or "Output" window.

5.4 ACTIVE TEMPLATE LIBRARY (ATL)

Active Template Library (ATL) was previously known as ActiveX Template Library). ATL is a Microsoft program library which is used by the developers for developing Active Server Page (ASP) and ActiveX program components with C++ (Visual C++ also).

A developer can forward user requests to a specific program in the Web server via common gateway interface application. In case the server is Microsoft's **Internet Information Server (IIS)**, the developers can insert a script in the HTML page. The page is known as **Active Server Page (ASP)**. The program script in the Active Server Page (ASP) is interpreted and processed in the server. After processing, the corresponding page is sent to the user.

The Active Template Library is used by the coders to develop Component Object Model (COM) object which can directly be spawned by the script on the ASP page. Objects which are developed using ATL consists of full controls, Internet Explorer controls, property pages and dialog boxes.

Active Template Library (ATL) is a collection of C++ classes. ATL enables the developers to develop Component Object Model (COM) objects. It consists of special support for primary COM features consisting of stock implementations, dual interfaces, and standard COM enumerator interfaces, point of connection, tear-off interfaces and ActiveX controls.

ATL programs are used to build single-threaded objects, apartment-model objects, free-threaded model objects or both free-threaded and apartment-model objects. ATL is intended to streamline the procedure of formulating effectual, accommodating, affluent controls.

The steps in this regard are as follows.

- Creation of the Project
- Providing Control to the Project
- Assigning Property to the Control
- Modifying the Control's Drawing Code

- Inserting an Event
- Inserting a Property Page
- Publishing the created Control on a Web Page

NOTES

5.4.1 Creation of the Project

Following are the steps for creating an ATL project by means of ATL project wizard.

a) For Visual Studio 2019: Select File '!New '! Project, type "atl" in the search box, and choose ATL Project.

b) Type Polygon as the project name. The source code can be found in \Users\<username>\source\repos

c) For Visual Studio 2019, the default values should be accepted. Click OK to proceed.

Now, the project will be created. Several files will get generated with this. These files can be viewed in Solution Explorer by expanding the Polygon object. The files are listed below:

- o Polygon.cpp
- o Polygon.def
- o Polygon.idl
- o Polygon.rgs
- o Polygon.rc
- o Resource.h
- o Polygonps.def
- o pch.cpp
- o pch.h

a) Go to "Solution Explorer" and right-click on the "Polygon project".

b) On the shortcut menu, click on "Properties".

c) Click on the Linker. Change the Per-UserRedirection option to "Yes".

d) Click "OK".

Adding a Control to the Project

A control shall be added in the created project. Then, we will build it and finally test it on a Web page.

For adding an object to the created ATL project, we need to follow the following steps:

a) Go to "Solution Explorer", right-click on the project name "Polygon" which is just created.

b) Go to "Add on the shortcut menu" and click "New Item" in the submenu.

a. The "Add New Item" dialog box gets displayed. Various object categories are provided in this dialog box.
c) Click on the "ATL" folder.
d) From the list of displayed templates on the right, select the "ATL Control". Then, click on "Add". By doing this the ATL Control wizard will get opened and the configuration of the control can be accomplished.
e) Type "PolyCtl" as the short name and other fields will get filled automatically. The process is not yet finished as additional changes are to be made now.

The ATL Control wizard's Names page contains the following fields:

- Short name
- Class
- .h file
- .cpp file
- CoClass
- Interface
- Type
- ProgID

This step is to allow support for rich error information and connection points. The following steps are to be performed:

a) Click "Options" for opening the "Options" page.
b) "Connection points" check box is to be selected. This option will build support for the outbound interface in the associated IDL file.

To extend the control's functionality,

a) Open the "Interfaces" page, click on "Interfaces".
b) "IProvideClassInfo2" should be selected. Next, Up arrow key should be used for moving it to the "Supported" list.
c) "ISpecifyPropertyPages" should be selected. Next, Up arrow key should be used for moving it to the "Supported" list.

The control can be made insertable i.e., the control can be embeddable into an application but the application must support embedded objects. Example of such application can be as MS Word or MS Excel.

For making the control insertable

a) "Appearance" should be clicked to open the "Appearance" page.
b) "Insertable" check box should be selected.

Need to add a "Fill Color" stock property.

Process to add a "Fill Color" stock property and creation of the control is as follows.

NOTES

a) "Stock Properties" should be for opening the "Stock Properties" page.
b) Under "Not supported", need to scroll down the list for probable stock properties. Next, "Fill Color" should be selected and the Up arrow should be clicked to move it to the "Supported" list.
c) Next, click on "Finish".

Various code gets changed and at the same time additional files are also created with these procedures. Following are the files created:

- o PolyCtl.h
- o PolyCtl.cpp
- o PolyCtl.rgs
- o PolyCtl.htm

The wizard changes to the following codes:

- Header files are added with #include statement. This includes required ATL files which are required to support the controls.
- Changes are made in "Polygon.idl". Changes consists of the inclusion of new control.
- The new control is added to the object map in "Polygon.cpp".

Now, the control can be built to experience the associated action.

Process of building and testing the Control

To implement this, Click "Build Polygon" on the "Build menu".

Once the control accomplishes the building task, "PolyCtl.htm" should be right-clicked in "Solution Explorer" and "View in Browser" should be selected. Only after this the HTML Web page consisting of the control is displayed on the screen. A page with the title "ATL 8.0 test page for object PolyCtl", and the associated control "PolyCtl" is also visible.

Adding a Property to the Control

Following are the steps which should be followed for adding the property definitions to the project.

a) Expand the Polygon branch in "Class View".
b) Right-click on the "IPolyCtl".
c) Click on "Add" on the "shortcut" menu, and then the "Add Property" should be clicked on. By doing this the "Add Property" wizard will be displayed.
d) "Sides" should be provided as the "Property Name".
e) "short" should be selected in the drop-down list of "Property Type".
f) Click on "OK" to complete the adding of property.
g) Polygon.idl should be opened from "Solution Explorer", and should be replaced with the following code at the end of the IPolyCtl : IDispatch interface.

```
short get_Sides();
void set_Sides(short value);
```

```
with
[propget, id(1), helpstring("property Sides")] HRESULT
Sides([out, retval] short *pVal);
[propput, id(1), helpstring("property Sides")] HRESULT
Sides([in] short newVal);
```

```
h)  PolyCtl.h should be opened from "Solution Explorer",
and the below mentioned code should be appended after the
definition of m_clrFillColor:
short m_nSides;
STDMETHOD(get_Sides)(short* pval);
STDMETHOD(put_Sides)(short newval);
```

Updation of the Get and Put Methods

a) The default value of m_nSides should be set. The default shape a triangle should be established by appending the below mentioned line to the constructor in PolyCtl.h:

```
m_nSides = 3;
```

b) Next step is the implementation of Get and Put methods. The function declarations of get_Sides and put_Sides have been appended to PolyCtl.h. To proceed further, the following code is for get_Sides and put_Sides should be appended to the PolyCtl.cpp

```
STDMETHODIMP CPolyCtl::get_Sides(short* pVal)
{
   *pVal = m_nSides;

   return S_OK;
}

STDMETHODIMP CPolyCtl::put_Sides(short newVal)
{
   if (2 < newVal && newVal < 101)
   {
      m_nSides = newVal;
      return S_OK;
   }
   else
   {
     return Error(_T("Shape must have between 3 and 100
sides"));
   }
}
```

The get_Sides method provides the present value of the "Sides" property through the "pVal" pointer. In the put_Sides method, the associated code guarantees the user is establishing the "Sides" property to a suitable value. The minimum value

should be 3 and the maximum value should be 100. This is because an array of pointers are used for each side.

Alteration of the Drawing Code

NOTES

By default, the control's illustration code exhibits a square and the associated text is PolyCtl.

In this step, the code needs to be changed and displaying something more fascinating. To implement this use the following steps.

- Alteration of the associated Header File
- Alteration of the "OnDraw" Function
- Adding a "Method" to Calculate the Polygon Points
- Initializing the "Fill Colour"

Alteration of the Associated Header File

This process should be imitated by introducing the math functions Sin and Cos. These are used for calculating the polygon points and formulating an array to store the coordinates.

Follow the steps given below for altering the header.

a) The following line should be inserted

```
#include <math.h> at the top of PolyCtl.h.
```

The upper part of the file should look like this:

```
#include <math.h>
#include "resource.h"      // main symbols
```

b) Implementation should be performed on the IProvideClassInfo interface to deliver method information for the control, by appending the following code to PolyCtl.h. Replace the below mentioned line In the CPolyCtl class:

```
public CComControl<CPolyCtl>
with
public CComControl<CPolyCtl>,
public  IProvideClassInfo2Impl<&CLSID_PolyCtl,
&DIID__IPolyCtlEvents, &LIBID_PolygonLib>
and in BEGIN_COM_MAP(CPolyCtl), append the following code:
COM_INTERFACE_ENTRY(IProvideClassInfo)
COM_INTERFACE_ENTRY(IProvideClassInfo2)
```

Once the polygon coordinates are calculated, it will be stored in an array of type POINT. Now, the array should be appended after the definition statement short m_nSides; in PolyCtl.h:

```
POINT m_arrPoint[100];
```

Alteration of the OnDraw Function

The OnDraw method should be modified in PolyCtl.h. The code which will be appended will create a new pen and brush by which the polygon can be drawn. The Ellipse and Polygon Win32 API functions should be called to perform the definite illustration.

NOTES

For modifying the OnDraw function, the existing OnDraw method in PolyCtl.h should be replaced with the following code:

```
HRESULT CPolyCtl::OnDraw(ATL_DRAWINFO& di)
{
   RECT& rc = *(RECT*)di.prcBounds;
   HDC hdc  = di.hdcDraw;

   COLORREF    colFore;
   HBRUSH      hOldBrush, hBrush;
   HPEN        hOldPen, hPen;

   // Translate m_colFore into a COLORREF type
   OleTranslateColor(m_clrFillColor, NULL, &colFore);

   // Create and select the colors to draw the circle
   hPen = (HPEN)GetStockObject(BLACK_PEN);
   hOldPen = (HPEN)SelectObject(hdc, hPen);
   hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
   hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

   Ellipse(hdc, rc.left, rc.top, rc.right, rc.bottom);

   // Create and select the brush that will be used to
fill the polygon
   hBrush    = CreateSolidBrush(colFore);
   SelectObject(hdc, hBrush);

   CalcPoints(rc);
   Polygon(hdc, &m_arrPoint[0], m_nSides);

   // Select back the old pen and brush and delete the
brush we created
   SelectObject(hdc, hOldPen);
   SelectObject(hdc, hOldBrush);
   DeleteObject(hBrush);

   return S_OK;
}
```

For adding the CalcPoints method, the declaration of CalcPoints should be added to the IPolyCtl public section of the CPolyCtl class in PolyCtl.h.

```
void CalcPoints(const RECT& rc);
```

NOTES

```
The end portion of the public section of the CPolyCtl
class should be as follows:
void FinalRelease()
   {
   }
public:
   void CalcPoints(const RECT& rc);

This implementation of the CalcPoints function should be
added at the bottom of PolyCtl.cpp

void CPolyCtl::CalcPoints(const RECT& rc)
{
   const double pi = 3.14159265358979;
   POINT   ptCenter;
   double  dblRadiusx = (rc.right - rc.left) / 2;
   double  dblRadiusy = (rc.bottom - rc.top) / 2;
   double  dblAngle = 3 * pi / 2;        // Start at the
top
   double  dblDiff  = 2 * pi / m_nSides;   // Angle each
side will make
   ptCenter.x = (rc.left + rc.right) / 2;
   ptCenter.y = (rc.top + rc.bottom) / 2;

   // Calculate the points for each side
   for (int i = 0; i < m_nSides; i++)
   {
     m_arrPoint[i].x = (long)(dblRadiusx * cos(dblAngle)
+ ptCenter.x + 0.5);
     m_arrPoint[i].y = (long)(dblRadiusy * sin(dblAngle)
+ ptCenter.y + 0.5);
      dblAngle += dblDiff;
   }
}
```

By default, green color should be used for initializing the fill color. This should be done by appending the following code to the CPolyCtl constructor in PolyCtl.h.

```
m_clrFillColor = RGB(0, 0xFF, 0);
The constructor should be similar as follows:

CPolyCtl()
{
   m_nSides = 3;
   m_clrFillColor = RGB(0, 0xFF, 0);
}
```

While rebuilding the control, it has to be ensured that the PolyCtl.htm file is closed. In case, it is open "Build Polygon" should be clicked on the "Build" menu. The control can once again be viewed from the PolyCtl.htm page, but ideally the ActiveX Control Test Container should be used for viewing it.

Building and Starting the ActiveX Control Test Container

a) On the Edit menu, in the "Test Container", click on "Insert New Control".

b) The control should be located, which will be called as PolyCtl class, and should be clicked on "OK". A green triangle inside a circle should be seen.

c) For modifying the properties on a dual interface from inside Test Container, "Invoke Methods" should be used.

For modifying a control's property inside the Test Container the follow the steps given below.

a) In Test Container, "Invoke Methods" should be clicked on the "Control" menu. The "Invoke Method" dialog box should be visible.

b) "PropPut" version of the Sides property should be selected from the "Method Name" drop-down list box.

c) Parameter Value box should have 5.

d) "Set Value" should be clicked and the "Invoke" should be clicked.

For adding a call to FireViewChange, the following process should be completed. Updation of the PolyCtl.cpp should be done by adding the call to FireViewChange to the put_Sides method. Once this step is completed, the put_Sides method will be as follows:

```
STDMETHODIMP CPolyCtl::put_Sides(short newVal)
{
   if (2 < newVal && newVal < 101)
   {
      m_nSides = newVal;
      FireViewChange();
      return S_OK;
   }
   else
   {
     return Error(_T("Shape must have between 3 and 100
sides"));
   }
}
```

On adding FireViewChange, rebuild should be done and then the control should be tried once more in the ActiveX Control Test Container. When the number of sides are changed and clicked on "Invoke", it will be observed that the control will change on immediate basis.

NOTES

Adding an Event

In this step, ClickIn and ClickOut event shall be added to the ATL control. The ClickIn event will be triggered in case the user clicks within the polygon and trigger ClickOut in case the user clicks outside. The steps to add an event are as follows:

- Adding of the ClickIn and ClickOut methods
- Generation of the Type Library
- Implementation of the Connection Point Interfaces

Following are the steps for adding the ClickIn and ClickOut methods.

In "Solution Explorer", Polygon.idl should be opened and the following code should be added under methods: in the dispInterface_IPolyCtlEvents declaration of the PolygonLib library:

```
[id(1), helpstring("method ClickIn")] void ClickIn([in]
LONG x,[in] LONG y);
[id(2), helpstring("method ClickOut")] void ClickOut([in]
LONG x,[in] LONG y);
```

For generating the type library, either the project should be rebuild or Polygon.idl file should be Right-click in the "Solution Explorer" and then "Compile" should be clicked which is available on the shortcut menu.

This process will create the Polygon.tlb file which is of the type library. The Polygon.tlb file is not visible from Solution Explorer because it is a binary file and cannot be viewed or edited directly.

Following are the step to implement the connection points.

In "Solution Explorer", open _IPolyCtlEvents_CP.h and append the following code under the public: statement in the CProxy_IPolyCtlEvents class.

```
VOID Fire_ClickIn(LONG x, LONG y)
{
    T* pT = static_cast<T*>(this);
    int nConnectionIndex;
    CComVariant* pvars = new CComVariant[2];
    int nConnections = m_vec.GetSize();

       for (nConnectionIndex = 0; nConnectionIndex <
nConnections; nConnectionIndex++)
     {
        pT->Lock();
                    CComPtr<IUnknown>   sp   =
m_vec.GetAt(nConnectionIndex);
        pT->Unlock();
                    IDispatch*   pDispatch   =
reinterpret_cast<IDispatch*>(sp.p);
        if (pDispatch != NULL)
         {
            pvars[1].vt = VT_I4;
            pvars[1].lVal = x;
```

NOTES

```
            pvars[0].vt = VT_I4;
            pvars[0].lVal = y;
            DISPPARAMS disp = { pvars, NULL, 2, 0 };
                    pDispatch->Invoke(0x1, IID_NULL,
LOCALE_USER_DEFAULT, DISPATCH_METHOD, &disp, NULL, NULL,
NULL);
        }
    }
    delete[] pvars;

}
VOID Fire_ClickOut(LONG x, LONG y)
{
    T* pT = static_cast<T*>(this);
    int nConnectionIndex;
    CComVariant* pvars = new CComVariant[2];
    int nConnections = m_vec.GetSize();

      for (nConnectionIndex = 0; nConnectionIndex <
nConnections; nConnectionIndex++)
    {
        pT->Lock();
                    CComPtr<IUnknown> sp =
m_vec.GetAt(nConnectionIndex);
        pT->Unlock();
                    IDispatch* pDispatch =
reinterpret_cast<IDispatch*>(sp.p);
        if (pDispatch != NULL)
        {
            pvars[1].vt = VT_I4;
            pvars[1].lVal = x;
            pvars[0].vt = VT_I4;
            pvars[0].lVal = y;
            DISPPARAMS disp = { pvars, NULL, 2, 0 };
                    pDispatch->Invoke(0x2, IID_NULL,
LOCALE_USER_DEFAULT, DISPATCH_METHOD, &disp, NULL, NULL,
NULL);
        }
    }
    delete[] pvars;

}
```

This file consists of a class named CProxy_IPolyCtlEvents which is derived from IConnectionPointImpl. _IPolyCtlEvents_CP.h that describes the two methods i.e. Fire_ClickIn and Fire_ClickOut. These methods accepts the two coordinate

parameters. These methods will be called when an event will be triggered from the control.

NOTES

After the creation of the control through "Connection points" option selected, the _IPolyCtlEvents_CP.h is generated. It has also appended CProxy_PolyEvents and IConnectionPointContainerImpl to the control's various inheritance list and enabled IConnectionPointContainer by appending suitable entries to the COM map.

Adding a Property Page

Property pages are organized as dispersed COM objects. This dispersed feature permits them to be shared when required. Following are the steps for adding a property page to the control.

- Create the property page resource
- Add codes appropriate for creating and managing the property page
- Add the property page to the control

Process for Adding a Property Page

a) Right click on the Polygon in "Solution Explorer".

b) Click Add '! New Item, on the shortcut menu.

c) From the list of templates, select ATL '! ATL "Property Page" and click "Add".

d) Enter "PolyProp" as the short name, once the "ATL Property Page Wizard" is displayed.

e) Click "Strings" to open the "Strings page" and enter "&Polygon" as the "Title".

At this stage, it is not necessary to generate a "Help file". Hence, delete the entry in that text box.

f) Click "Finish" to create the property page object.

Following are the three files which will be created.

- PolyProp.h
- PolyProp.cpp
- PolyProp.rgs

The following changes in the program code can be observed.

- The new "Property Page" gets appended to the object entry map in "Polygon.cpp".
- The "PolyProp" class is appended to the "Polygon.idl" file.
- The new registry script file "PolyProp.rgs" is appended to the "project resource".
- A "dialog box template" is added to the project resource for the specific property page.
- The "property strings" which was specified earlier are appended to the "resource string table".

- Now, based on the wish list, the required fields have to be added on the property page.

Adding Fields to the Property Page

a) Double-click the "Polygon.rc" resource file which will be found in the "Solution Explorer". It will open "Resource View".

b) Expand the Dialog node in the "Resource View" and double-click "IDD_POLYPROP". The dialog box which gets displayed is blank apart form a label that asks to insert the created controls.

c) The appropriate label should be selected and to be changed to read "Sides:" by modifying the "Caption" text in the "Properties" window.

d) The label box should be resized, so that it is able to accept the text size.

e) "Edit Control" should be dragged from the Toolbox" to the right side of the label.

f) The ID of the edit control should be changed to "IDC_SIDES" through the "Properties" window.

It will complete the procedure of creating the property page resource.

Follow the steps given below to alter the "Apply" function for setting the number of sides.

"Apply" function in PolyProp.h should be replaced with the following code.

```
STDMETHOD(Apply)(void)
{
  USES_CONVERSION;
  ATLTRACE(_T("CPolyProp::Apply\n"));
  for (UINT i = 0; i < m_nObjects; i++)
  {
        CComQIPtr<IPolyCtl, &IID_IPolyCtl>
pPoly(m_ppUnk[i]);
     short nSides = (short)GetDlgItemInt(IDC_SIDES);
     if FAILED(pPoly->put_Sides(nSides))
      {
        CComPtr<IErrorInfo> pError;
        CComBSTR strError;
        GetErrorInfo(0, &pError);
        pError->GetDescription(&strError);
              MessageBox(OLE2T(strError), _T("Error"),
MB_ICONEXCLAMATION);
        return E_FAIL;
      }
  }
  m_bDirty = FALSE;
  return S_OK;
}
```

NOTES

The property page's dirty flag also needs to be set. This needs to be set for indicating the "Apply" button to be enabled. This takes place when a user alters the value in the "Sides" edit box.

Following are the steps to handle the "Apply" button.

a) Right-click on "CPolyProp" in "Class View" and click "Properties" on the "shortcut" menu.
b) In the "Properties" window, click on the "Events" icon.
c) In the event list, expand the "IDC_SIDES" node.
d) "EN_CHANGE" should be selected and from the drop-down menu to the right. Click on <Add> "OnEnChangeSides". The declaration of "OnEnChangeSides" handler shall be appended to "Polyprop.h" and the handler implementation to "Polyprop.cpp".

Next, the handler needs to be modified. Following are the steps for modifying the OnEnChangeSides method.

Following code should be added in the "Polyprop.cpp" to the "OnEnChangeSides" method.

```
LRESULT CPolyProp::OnEnChangeSides(WORD /*wNotifyCode*/
, WORD /*wID*/,
  HWND /*hWndCtl*/, BOOL& /*bHandled*/)
{
  SetDirty(TRUE);

  return 0;
}
```

"OnEnChangeSides" will be called when a "WM_COMMAND" message will be sent with the "EN_CHANGE" notification for the "IDC_SIDES" control. "OnEnChangeSides" then calls "SetDirty" and passes "TRUE" to affirm that the property page is now dirty and "Apply" button must be enabled.

Following are the steps for adding the property page.
Open the "PolyCtl.h" and add the following lines to the property map.

```
PROP_ENTRY_TYPE("Sides", 1, CLSID_PolyProp, VT_INT)
PROP_PAGE(CLSID_PolyProp)
```

The control's property map will be as follows:

```
BEGIN_PROP_MAP(CPolyCtl)
  PROP_DATA_ENTRY("_cx", m_sizeExtent.cx, VT_UI4)
  PROP_DATA_ENTRY("_cy", m_sizeExtent.cy, VT_UI4)
#ifndef _WIN32_WCE
    PROP_ENTRY_TYPE("FillColor",  DISPID_FILLCOLOR,
CLSID_StockColorPage, VT_UI4)
#endif
  PROP_ENTRY_TYPE("Sides", 1, CLSID_PolyProp, VT_INT)
  PROP_PAGE(CLSID_PolyProp)
  // Example entries
```

```
    // PROP_ENTRY("Property Description", dispid, clsid)
    // PROP_PAGE(CLSID_StockColorPage)
END_PROP_MAP()
```

NOTES

A "PROP_PAGE" macro should be added with the "CLSID" of the property page. But, in case the PROP_ENTRY macro is used as specified, the "Sides" property value will also get saved at the time when the control is saved.

Following are the steps for building and testing the control.

The control should be built and inserted into ActiveX Control "Test" Container. In the Edit menu, click PolyCtl Class Object in the "Test" Container. The property page is displayed along with the information which was added.

The state of the "Apply" button is currently in disabled state. Once a value is inserted in the "Sides" box "Apply" button becomes activated. After entering some value, click on the "Apply" button. The associated control display gets changed and the "Apply" button is once more in disabled state. An associated error message will pop up in case invalid value is entered. This is the error message which was set from the "put_Sides" function.

Placing the Control on the Web Page

The control is now ready and it is not required to set the control working in a Web page. An HTML file consists of the control was created at the time when the control was defined. The "PolyCtl.htm" file can be opened from "Solution Explorer" and the control can be seen on the web page.

Now, the functionality to the control should be added and the Web page should be scripted so that it responds to events. The control should also be modified to permit internet explorer be aware of that the control is safe for scripting.

Following are the steps to add control features.

Open the "PolyCtl.cpp" and replace the content with the following code:

```
if (PtInRegion(hRgn, xPos, yPos))
    Fire_ClickIn(xPos, yPos);
else
    Fire_ClickOut(xPos, yPos);

with

short temp = m_nSides;
if (PtInRegion(hRgn, xPos, yPos))
{
    Fire_ClickIn(xPos, yPos);
    put_Sides(++temp);
}
else
{
    Fire_ClickOut(xPos, yPos);
```

```
    put_Sides(—temp);
}
```

NOTES

After this, the shape will now add or remove sides on the basis of where it is clicked.

Apply the following steps to script the Web page.

a) Open "PolyCtl.htm" and select "HTML" view. Add the following code to the HTML file. These specified codes should be appended after </OBJECT> but before </BODY>.

```
<SCRIPT LANGUAGE="VBScript">
<!—
    Sub PolyCtl_ClickIn(x, y)
        MsgBox("Clicked (" & x & ", " & y & ") - adding side")
    End Sub
    Sub PolyCtl_ClickOut(x, y)
        MsgBox("Clicked (" & x & ", " & y & ") - removing side")
    End Sub
—>
</SCRIPT>
```

b) HTML file should be saved.

5.5 DATABASE APPLICATION

ActiveX Data Objects (ADO) is a set of application components which offers a programming interface to get connected with the data sources from other applications. ADO is considered to be a layer for accessing data in a common fashion through the program. It removes the requirement to own information of database deployment and diminishes, the intricacy of handling with the low-level code which is required to access the data.

ADO was initiated from the notion of RDO (Remote Data Object) and DAO (Data Access Object). One of the elements of MDAC (Microsoft Data Access Components), ADO and other MDAC elements delivers a framework of constituents referred by the client programs to access SQL, semi-structured and bequest data storages.

ADO.NET is an innovative data access technology which is fabricated to operate in the .NET atmosphere. It is founded on the detached model for data access. ADO refers to COM (Component Object Model) technology for offering

data access facility from unmanaged programs. ADO.NET depends on "managed providers" of the CLR (Common Language Runtime) of the .NET framework.

NOTES

OLEDB provider is used by ADO for establishing connectivity with the data storage for accessing relevant data. OLEDB is an element and which is a program-based interface. It is used for interaction with various data sources. These data sources can either be relational database or non-relational databases. Non-relational databases consist of object databases, web pages, spreadsheets or mail. Earlier to OLEDB and ADO, ODBC (Open Database Connectivity) was the most common framework used in applications for all platforms.

ADO provides significant characteristics in creating various client/server and Web-centric applications.

The object model of ADO consists of four collections of 12 objects. The various collections are fields, properties, parameters and errors. Each collection comprises of the following 12 objects:

i. **Connection** – Used for establishing connectivity with the data source via OLE DB.
ii. **Command** – Used for transporting directive (SQL query or stored procedure) to the data provider.
iii. **Recordset** - An array of records which is indicating the data.
iv. **Immediate** – It is a recordset which is locked in either positive or negative manner.
v. **Batch** – Used for database committing rollback.
vi. **Transaction** – Refers to the database transaction.
vii. **Record** – Represents a set of fields.
viii. **Stream** – Used for reading and writing a series of bytes.
ix. **Parameter** – Used for altering the functionality.
x. **Field** – Represents a column of a database.
xi. **Property** – It is the capability of OLEDB provider.
xii. **Error** – It represents the error experienced by the OLEDB provider at the time of its execution.

ADO 2.8 is the latest version and has the following features:

- **Components**: These are used for accessing the data and manipulating the data from various sources of data. It also provides assists like ease of use, quicker access, low memory overhead and minor disk trail.
- **ADO MD (MultiDImensional)**: This is used to get connected to multidimensional data i.e., CubeDef and CellSet objects. Like ADO, ADO MD also uses a primary OLE DB provider to acquire access to the data. To function with ADO MD, the provider should be a Multidimensional Data Provider (MDP) as stated by the OLE DB for OLAP requirement. MDPs furnishes the data in multidimensional forms

NOTES

which is just opposite to TDPs. Tabular Data Providers (TDPs) provide the data in a tabular form.

- **RDS (Remote Data Services)**: It is a feature of ADO. This is used to retrieve and update data in the server from a client using a single transaction. Data can be moved from the database server to an application used by the client or in various Web pages.
- **ADOX (ADO Extensions)**: Microsoft ActiveX Data Objects Extensions for Data Definition Language and Security (ADOX) is an expansion to the ADO objects and programming framework. This is a supplementary array of elements which are used to generate and maintain objects related to schema (i.e., tables or procedures) and implement security (for user and group). ADOX is consider to be a companion library to the core ADO objects.

5.5.1 ActiveX Data Objects (ADO)

ADO objects are used for the following activities.

- Using SQL, queries are generated to fetch data from the database and display the results.
- Used to access data from a file which is stored in the cloud.
- Used to alter messages and folders in an e-mail application.
- Used fetch data from a database and save it in a XML file.
- Execution of various commands defined with XML.
- Retrieve data from an XML stream.
- Saving data in a binary or XML stream.
- Permitting a user to see and alter data in database tables.
- Used to create and use commands pertaining to parameterized database.
- Execution of stored procedures.
- Creation of a flexible structure in a dynamic manner which is known as "Recordset".
- Accomplish operations related to transactional database.
- Applying filters and sort local version of database based on the run-time principles.
- Creation and manipulation of hierarchical consequences from databases.
- Mapping the fields of the database with the data-aware elements.
- Creation of distant, disconnected RecordSet.

ADO Object Model

Figure 5.4 show the ADO objects and their collections.

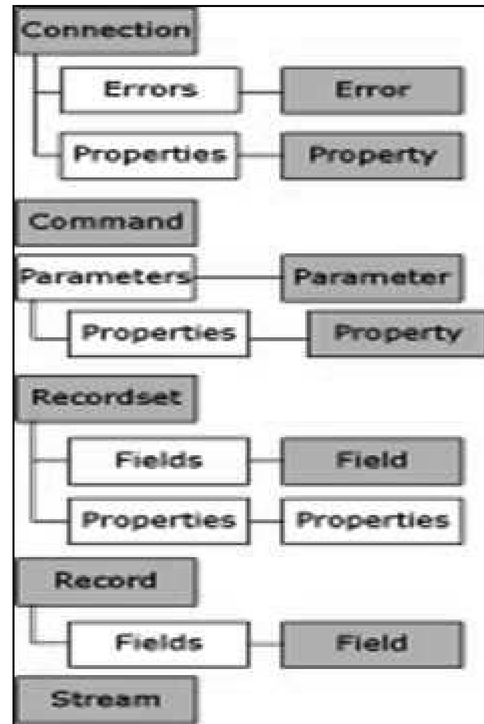


Fig. 5.4 *ADO Objects and their Collections*

ADO Collections

The relationships between the collections and the ADO objects is given in Figure 5.4.

Each collection consists of its corresponding object. For example, an Error object is contained in an Errors collection.

Collection	Description
Errors	Consists of all the error objects generated during a particular provider-related failure.
Fields	Consists of all the field objects of a Recordset object.
Parameters	Consists of all the parameter objects of a Command object.
Properties	Consists of all the property objects for a particular occurrence of an object.

ADO Dynamic Properties

In ADO, dynamic properties can be inserted to the properties collections of the Connection, Command, or Recordset objects. The primary source for these properties is either from a data provider (OLE DB Provider for SQL Server), or a service provider (Microsoft Cursor Service for OLE DB).

NOTES

Special functionality with ADO is provided in the following table.

Dynamic property	Description
Optimize	States, in case, an index should be formed on this field.
Prompt	States, in case, the OLE DB provider should ask for initialization information to the user.
Reshape Name	States a name for the Recordset object.
Resync Command	States a command string provided by the user that the Resync method provided to refresh the data in the table specified in the Unique Table dynamic property.
Unique Table, Unique Schema, Unique Catalog	**Unique Table** States the name of the base table on which updates, insertions, and deletions related operations are permitted. **Unique Schema** States the schema or name of the owner of the table. **Unique Catalog** States the catalogue or name of the database which consists of the table.
Update Resync	States in case the UpdateBatch method is maintained by an implicit Resync method operation. If it is maintained, it defines the opportunity of that specific operation.

"Connection" object or "Recordset" object are used to establish connectivity with a data source.

"Connection" object

A "Connection String" comprises of a set of argument/value pairs which are divided by semi-colons, with the values bounded within single quotes. Following in an example of "connection string".

```
Dim sConn As String
sConn = "Provider='SQLOLEDB';Data Source='MySqlServer';"
& _
              "Initial Catalog='Northwind';Integrated
Security='SSPI';"
```

"Recordset" Object

Alternatively, Recordset.Open can be used to indirectly create a connectivity and provide required commands using this connection in a sole operation. Consider an example as follows:

```
Dim oRs As ADODB.Recordset
Dim sConn As String
Dim sSQL as String

sConn = "Provider='SQLOLEDB';Data Source='MySqlServer';"
& _           "Initial Catalog='Northwind';Integrated
Security='SSPI';"

sSQL = "SELECT ProductID, ProductName, CategoryID,
UnitPrice " & _
```

```
            "FROM Products"

'Create and Open the Recordset object.
Set oRs = New ADODB.Recordset
oRs.CursorLocation = adUseClient
oRs.Open sSQL, sConn, adOpenStatic, _
adLockBatchOptimistic, adCmdText

MsgBox oRs.RecordCount

oRs.MarshalOptions = adMarshalModifiedOnly
'Disconnect the Recordset.
Set oRs.ActiveConnection = Nothing
oRs.Close
Set oRs = Nothing
```

NOTES

5.5.2 Database Application using ADO

The following code illustrates a sample program named HelloData in ADO. The code below provides simple application to perform the foremost ADO operations i.e., getting, examining, editing, and updating data. These operations are implmented against the Northwind sample database included with Microsoft® SQL Server.

```
'BeginHelloData
Option Explicit

Dim m_oRecordset As ADODB.Recordset
Dim m_sConnStr As String
Dim m_flgPriceUpdated As Boolean

Private Sub cmdGetData_Click()
    GetData

    If Not m_oRecordset Is Nothing Then
        If m_oRecordset.State = adStateOpen Then
            'Set the proper states for the buttons.
            cmdGetData.Enabled = False
            cmdExamineData.Enabled = True
        End If
    End If
End Sub

Private Sub cmdExamineData_Click()
    ExamineData
```

NOTES

```
End Sub

Private Sub cmdEditData_Click()
    EditData
End Sub

Private Sub cmdUpdateData_Click()
    UpdateData

    'Set the proper states for the buttons.
    cmdUpdateData.Enabled = False
End Sub

Private Sub GetData()
    On Error GoTo GetDataError

    Dim sSQL As String
    Dim oConnection1 As ADODB.Connection

        m_sConnStr = "Provider='SQLOLEDB';Data
Source='MySqlServer';" & _
             "Initial Catalog='Northwind';Integrated
Security='SSPI';"

    'Create and Open the Connection object.
    Set oConnection1 = New ADODB.Connection
    oConnection1.CursorLocation = adUseClient
    oConnection1.Open m_sConnStr

    sSQL = "SELECT ProductID, ProductName, CategoryID,
UnitPrice " & _
             "FROM Products"

    'Create and Open the Recordset object.
    Set m_oRecordset = New ADODB.Recordset
    m_oRecordset.Open sSQL, oConnection1, adOpenStatic,
_
            adLockBatchOptimistic, adCmdText

    m_oRecordset.MarshalOptions = adMarshalModifiedOnly

    'Disconnect the Recordset.
    Set m_oRecordset.ActiveConnection = Nothing
    oConnection1.Close
    Set oConnection1 = Nothing
```

NOTES

```
    'Bind Recordset to the DataGrid for display.
    Set grdDisplay1.DataSource = m_oRecordset

    Exit Sub

GetDataError:
    If Err <> 0 Then
        If oConnection1 is Nothing Then
                    HandleErrs "GetData",
m_oRecordset.ActiveConnection
        Else
          HandleErrs "GetData", oConnection1
        End If
    End If

    If Not oConnection1 Is Nothing Then
            If oConnection1.State = adStateOpen Then
oConnection1.Close
        Set oConnection1 = Nothing
    End If
End Sub

Private Sub ExamineData()
    On Err GoTo ExamineDataErr

    Dim iNumRecords As Integer
    Dim vBookmark As Variant

    iNumRecords = m_oRecordset.RecordCount

    DisplayMsg "There are " & CStr(iNumRecords) & _
             " records in the current Recordset."

    'Loop through the Recordset and print the
    'value of the AbsolutePosition property.
    DisplayMsg "****** Start AbsolutePosition Loop ******"

    Do While Not m_oRecordset.EOF
        'Store the bookmark for the 3rd record,
        'for demo purposes.
        If m_oRecordset.AbsolutePosition = 3 Then _
            vBookmark = m_oRecordset.Bookmark

        DisplayMsg m_oRecordset.AbsolutePosition
```

NOTES

```
        m_oRecordset.MoveNext
    Loop
   DisplayMsg "****** End AbsolutePosition Loop ******"
& vbCrLf

    'Use our bookmark to move back to 3rd record.
   m_oRecordset.Bookmark = vBookmark
    MsgBox vbCr & "Moved back to position" & _
               m_oRecordset.AbsolutePosition & " using
bookmark.", , _
            "Hello Data"

    'Display meta-data about each field. See WalkFields()
sub.
    Call WalkFields

    'Apply a filter on the type field.
   MsgBox "Filtering on type field. (CategoryID=2)", _
            vbOKOnly, "Hello Data"

   m_oRecordset.Filter = "CategoryID=2"

    'Set the proper states for the buttons.
   cmdExamineData.Enabled = False
   cmdEditData.Enabled = True

    Exit Sub

ExamineDataErr:
                   HandleErrs    "ExamineData",
m_oRecordset.ActiveConnection
End Sub

Private Sub EditData()
    On Error GoTo EditDataErr

    'Recordset still filtered on CategoryID=2.
    'Increase price by 10% for filtered records.
    MsgBox "Increasing unit price by 10%" & vbCr & _
       "for all records with CategoryID = 2.", , "Hello
Data"

   m_oRecordset.MoveFirst

    Dim cVal As Currency
```

NOTES

```
    Do While Not m_oRecordset.EOF
       cVal = m_oRecordset.Fields("UnitPrice").Value
       m_oRecordset.Fields("UnitPrice").Value = (cVal *
1.1)
        m_oRecordset.MoveNext
    Loop

    'Set the proper states for the buttons.
    cmdEditData.Enabled = False
    cmdUpdateData.Enabled = True

    Exit Sub

EditDataErr:
   HandleErrs "EditData", m_oRecordset.ActiveConnection
End Sub

Private Sub UpdateData()
    On Error GoTo UpdateDataErr

    Dim oConnection2 As New ADODB.Connection

   MsgBox "Removing Filter (adFilterNone).", "Hello Data"
   m_oRecordset.Filter = adFilterNone

   Set grdDisplay1.DataSource = Nothing
   Set grdDisplay1.DataSource = m_oRecordset

   MsgBox "Applying Filter (adFilterPendingRecords).",
"Hello Data"
   m_oRecordset.Filter = adFilterPendingRecords

   Set grdDisplay1.DataSource = Nothing
   Set grdDisplay1.DataSource = m_oRecordset

   DisplayMsg "*** PRE-UpdateBatch values for 'UnitPrice'
field. ***"

    'Display Value, UnderlyingValue, and OriginalValue
for
    'type field in first record.
   If m_oRecordset.Supports(adMovePrevious) Then
       m_oRecordset.MoveFirst
        DisplayMsg "OriginalValue   = " & _
      m_oRecordset.Fields("UnitPrice").OriginalValue
```

```
        DisplayMsg "Value           = " & _
      m_oRecordset.Fields("UnitPrice").Value
    End If

     oConnection2.ConnectionString = m_sConnStr
     oConnection2.Open

     Set m_oRecordset.ActiveConnection = oConnection2
     m_oRecordset.UpdateBatch

     m_flgPriceUpdated = True

         DisplayMsg "*** POST-UpdateBatch values for
'UnitPrice' field ***"

    If m_oRecordset.Supports(adMovePrevious) Then
        m_oRecordset.MoveFirst
        DisplayMsg "OriginalValue   = " & _
         m_oRecordset.Fields("UnitPrice").OriginalValue
        DisplayMsg "Value           = " & _
          m_oRecordset.Fields("UnitPrice").Value
    End If

    MsgBox "See value comparisons in txtDisplay.", _
          "Hello Data"

    'Clean up
    oConnection2.Close
    Set oConnection2 = Nothing
    Exit Sub

UpdateDataErr:
    If Err <> 0 Then
       HandleErrs "UpdateData", oConnection2
    End If

    If Not oConnection2 Is Nothing Then
            If oConnection2.State = adStateOpen Then
oConnection2.Close
       Set oConnection2 = Nothing
    End If
End Sub

Private Sub WalkFields()
    On Error GoTo WalkFieldsErr
```

NOTES

NOTES

```
    Dim iFldCnt As Integer
    Dim oFields As ADODB.Fields
    Dim oField As ADODB.Field
    Dim sMsg As String

    Set oFields = m_oRecordset.Fields

    DisplayMsg "****** BEGIN FIELDS WALK ******"

    For iFldCnt = 0 To (oFields.Count - 1)
        Set oField = oFields(iFldCnt)
        sMsg = ""
        sMsg = sMsg & oField.Name
              sMsg = sMsg & vbTab & "Type: " &
GetTypeAsString(oField.Type)
           sMsg = sMsg & vbTab & "Defined Size: " &
oField.DefinedSize
           sMsg = sMsg & vbTab & "Actual Size: " &
oField.ActualSize

        grdDisplay1.SelStartCol = iFldCnt
        grdDisplay1.SelEndCol = iFldCnt
        DisplayMsg sMsg
        MsgBox sMsg, "Hello Data"
    Next iFldCnt

    DisplayMsg "****** END FIELDS WALK ******" & vbCrLf

    'Clean up
    Set oField = Nothing
    Set oFields = Nothing
    Exit Sub

WalkFieldsErr:
    Set oField = Nothing
    Set oFields = Nothing

    If Err <> 0 Then
      MsgBox Err.Source & "—>" & Err.Description, "Error"
    End If
End Sub

Private Function GetTypeAsString(dtType As
ADODB.DataTypeEnum) As String
```

NOTES

```
    'To save space, we are only checking for data types
    'that we know are present.
    Select Case dtType
        Case adChar
            GetTypeAsString = "adChar"
        Case adVarChar
            GetTypeAsString = "adVarChar"
        Case adVarWChar
            GetTypeAsString = "adVarWChar"
        Case adCurrency
            GetTypeAsString = "adCurrency"
        Case adInteger
            GetTypeAsString = "adInteger"
    End Select
End Function

Private Sub HandleErrs(sSource As String, ByRef
m_oConnection As ADODB.Connection)
    DisplayMsg "ADO (OLE) ERROR IN" & sSource
    DisplayMsg vbTab & "Error: " & Err.Number
    DisplayMsg vbTab & "Description: " & Err.Description
    DisplayMsg vbTab & "Source: " & Err.Source

    If Not m_oConnection Is Nothing Then
        If m_oConnection.Errors.Count <> 0 Then
            DisplayMsg "PROVIDER ERROR"
            Dim oError1 As ADODB.Error
            For Each oError1 in m_oConnection.Errors
                DisplayMsg vbTab & "Error: " &
oError1.Number
                DisplayMsg vbTab & "Description: " &
oError1.Description
                DisplayMsg vbTab & "Source: " &
oError1.Source
                DisplayMsg vbTab & "Native Error:" &
oError1.NativeError
                DisplayMsg vbTab & "SQL State: " &
oError1.SQLState
            Next oError1
            m_oConnection.Errors.Clear
            Set oError1 = Nothing
        End If
    End If

    MsgBox "Error(s) occurred. See txtDisplay1 for specific
information.", _
```

```
          “Hello Data”

    Err.Clear
End Sub

Private Sub DisplayMsg(sText As String)
   txtDisplay1.Text = (txtDisplay1.Text & vbCrLf & sText)
End Sub

Private Sub Form_Resize()
     grdDisplay1.Move 100, 700, Me.ScaleWidth - 200,
(Me.ScaleHeight - 800) / 2
      txtDisplay1.Move 100, grdDisplay1.Top +
grdDisplay1.Height + 100, _
                  Me.ScaleWidth - 200, (Me.ScaleHeight
- 1000) / 2
End Sub

Private Sub Form_Load()
   cmdGetData.Enabled = True
   cmdExamineData.Enabled = False
   cmdEditData.Enabled = False
   cmdUpdateData.Enabled = False

   grdDisplay1.AllowAddNew = False
   grdDisplay1.AllowDelete = False
   grdDisplay1.AllowUpdate = False
   m_flgPriceUpdated = False
End Sub

Private Sub Form_Unload(Cancel As Integer)
   On Error GoTo ErrHandler:

   Dim oConnection3 As New ADODB.Connection
   Dim sSQL As String
   Dim lAffected As Long

   ‘Undo the changes we’ve made to the database on the
server.
   If m_flgPriceUpdated Then
      sSQL = “UPDATE Products SET UnitPrice=(UnitPrice/
1.1) “ & _
           “WHERE CategoryID=2”
       oConnection3.Open m_sConnStr
       oConnection3.Execute sSQL, lAffected, adCmdText
```

NOTES

```
        MsgBox "Restored prices for" & CStr(lAffected) & _
              "records affected.", , "Hello Data"
    End If

    'Clean up
    oConnection3.Close
    Set oConnection3 = Nothing
    m_oRecordset.Close
    Set m_oRecordset = Nothing
    Exit Sub

ErrHandler:

    If Not oConnection3 Is Nothing Then
            If oConnection3.State = adStateOpen Then
oConnection3.Close
        Set oConnection3 = Nothing
    End If
    If Not m_oRecordset Is Nothing Then
            If m_oRecordset.State = adStateOpen Then
m_oRecordset.Close
        Set m_oRecordset = Nothing
    End If
End Sub

'EndHelloData
```

NOTES

Check Your Progress

7. What custom build steps?
8. Define custom build tools.
9. What is ATL?
10. Write a note on OnDraw method?
11. What are ActiveX data objects?

5.6 ANSWER TO 'CHECK YOUR PROGRESS'

1. A socket is considered to be an endpoint of a bi-directional communication between two applications running on a network. A socket is associated to a specific port, so that the TCP layer can recognize the program and send the data throught the socket.
2. Technique in Client-Server Communication are as follows:
   - Socket: Establishes a new communication

- Bind: Attaches a local address to a specified socket
- Listen: Publicizes the preparedness to receive connections
- Accept: Impede caller that specific time when a connection invitation reaches
- Connect: Aggressively try to initiate a connection
- Send: Send data by the established connection
- Receive: Receive data by the established connection
- Close: Established connection is released

3. The fundamental steps for client-server setup are as follows:
   - A client application sends a service request to a server application.
   - The server application sends a acknowledgement.
   - The connection is established.
   - Some of the rudimentary data communications between client and server.
4. A port is a communication endpoint. At the software level, a port is a logical construct that identifies a specific process or a type of network service. A port is identified for each transport protocol and address combination by a 16-bit unsigned number, known as the port number. The most common transport protocols that use port numbers are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).
5. A socket is a combination of ports and IP addresses. An Internet Protocol address (IP address) is the logical address of our network hardware by which other devices identify it in a network.
6. MAPI is a Microsoft Windows program interface which is used to send mails across windows application and documents can be attached within the mails. MAPI consists of applications like word processors, spreadsheets, presentation and graphics applications. All MAPI-compatible applications consist of a Send Mail which is used to send mails.
7. A "Custom Build Step" can mention command lines for execution, any supplementary input or output files and a message for displaying.
8. Custom build tools is a build rule which is related to 1 or more files. A "custom build" step can distribute input files to a "custom build tool" and the consequence is one or more output files. For example, the help files in an MFC are developed using custom build tool.
9. Active Template Library (ATL) was previously known as ActiveX Template Library). ATL is a Microsoft program library which is used by the developers for developing Active Server Page (ASP) and ActiveX program components with C++ (Visual C++ also).
10. The OnDraw method should be modified in PolyCtl.h. The code which will be appended will create a new pen and brush by which the polygon can be drawn and then the Ellipse and Polygon Win32 API functions should be called to perform the definite illustration.

NOTES

11. ActiveX Data Objects (ADO) is a set of application components which offers a programming interface to get connected with the data sources from other applications. ADO is considered to be a layer for accessing data in a common fashion via the program. It removes the requirement to own the information of database deployment and diminishes the intricacy of handling with the low-level code which is required to access the data.

5.7 SUMMARY

- A socket is considered to be an endpoint of a bi-directional communication between two applications running on a network. A socket is associated to a specific port so that the TCP layer can recognize the program and send the data via the socket.
- In C++, Socket programming is the method of linking two nodes (ideally SNMP enabled nodes) over a network so that the communication can take place at ease without any loss of data.
- Post the socket is created, it is essential to establish a method of receiving inputs from the users. This "Input Stream" function will return the "InputStream" which will enable to associate the data to this socket. This will also generate exceptions as required.
- Post the socket is created, it is required to get an output from the user. This "output stream" function will return the "OutputStream" which will enable to associate the data to this socket.
- Post the socket is created, it is essential to close it as it can't be kept opened. This function will close the socket.
- A port is a communication endpoint. At the software level, within an operating system, a port is a logical construct that identifies a specific process or a type of network service. A port is identified for each transport protocol and address combination by a 16-bit unsigned number, known as the port number. The most common transport protocols that use port numbers are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).
- IP address is the logical address of our network hadware by which other devices identify it in a network.
- MAPI stands for Messaging Application Program Interface. From the name itself we can understand that this has something to do with mail. MAPI is a Microsoft Windows program interface which is used to send mails across windows application and documents can be attached within the mails.
- MAPI consists of applications like word processors, spreadsheets, presentation and graphics applications. All MAPI-compatible applications consist of a Send Mail which is used to send mails.
- MAPI references re written for C and C++ developers with a variation of requirements and knowledge with messaging. (MAPI) is a widespread collection of functions that is used by the developers during developing mail-enabled applications.

- Custom build steps is a build rule which is related with a project. A "Custom Build Step" can mention command lines for execution, any supplementary input or output files and a message for displaying.
- Custom build tools is a build rule which is related to 1 or more files. A "custom build" step can distribute input files to a "custom build tool" and the consequence is one or more output files. For example, the help files in an MFC are developed using custom build tool.
- Active Template Library (ATL) was previously known as ActiveX Template Library). ATL is a Microsoft program library which is used by the developers for developing Active Server Page (ASP) and ActiveX program components with C++ (Visual C++ also).
- The OnDraw method should be modified in PolyCtl.h. The code which will be appended will create a new pen and brush by which the polygon can be drawn and then the Ellipse and Polygon Win32 API functions should be called to perform the definite illustration.
- ActiveX Data Objects (ADO) is a set of application components which offers a programming interface to get connected with the data sources from other applications. ADO is considered to be a layer for accessing data in a common fashion via the program. It removes the requirement to own the information of database deployment and diminishes the intricacy of handling with the low-level code which is required to access the data.
- A "Connection String" comprises of a set of argument/value pairs which are divided by semi-colons, with the values bounded within single quotes.

5.8 KEY TERMS

- **Socket:** A socket is considered to be an endpoint of a bi-directional communication between two applications running on a network. A socket is associated to a specific port, so that the TCP layer can recognize the program and send the data through the socket.
- **Port:** A port is a communication endpoint. At the software level, a port is a logical construct that identifies a specific process or a type of network service. A port is identified for each transport protocol and address combination by a 16-bit unsigned number, known as the port number. The most common transport protocols that use port numbers are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).
- **Custom Build Tools:** It is a build rule which is related to one or more files. A "Custom Build" step can distribute input files to a "custom build tool" and the consequence is one or more output files. For example, the help files in an MFC are developed using custom build tool.
- **Active Template Library (ATL):** It was previously known as ActiveX Template Library). ATL is a Microsoft program library which is used by the developers for developing Active Server Page (ASP) and ActiveX program components with C++ and Visual C++.

NOTES

- **ActiveX Data Objects:** It is a set of application components which offers a programming interface to get connected with the data sources from other applications. ADO is considered to be a layer for accessing data in a common fashion via the program. It removes the requirement to own the information of database deployment and diminishes the intricacy of handling with the low-level code which is required to access the data.
- **Connection String:** A "Connection String" comprises of a set of arguments which are separated by semi-colons and with the values bounded within single quotes.

5.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is socket?
2. What do you mean by the port, MAPI and addresses?
3. What are the commonly used ports associated with socket?
4. What are the feature of MAPI?
5. What are the benefits of MAPI over HTTP?
6. Define active template library.
7. What are Activex Data Objects (ADO)?

Long-Answer Questions

1. Discuss about the socket, MAPI and the internet with the help of examples.
2. How will you create a socket program? Give appropriate example.
3. Elaborate on the active template library with the help of relevant examples.
4. Create a database application using ADO.

5.10 FURTHER READING

Cornell, Gary. 1998. *Visual Basic 6 from the Ground Up*. New Delhi: Tata McGraw-Hill.

Manchanda, Mahesh. 2009. *Visual Programming*. New Delhi: Vikas Publishing House Pvt. Ltd.

Balena, Francesco. 1999. *Programming Microsoft Visual Basic 6.0*. Bangalore: WP Publishers and Distributors (P) Ltd.

Petroutsos, Evangelos. 1998. *Mastering Visual Basic 6*, 1st Edition. New Delhi: BPB Publications.

Deitel, Harvey M., Paul J. Deitel and T. Tem R. Nieto. 1999. *Visual Basic 6: How to Program*. New Jersey: Prentice-Hall.

Donald, Bob and Oancea Gabriel. 1999. *Visual Basic 6 from Scratch*. New Delhi: Prentice-Hall of India.