



॥ सरस्वती नः सुभगा मयस्करत् ॥

Uttar Pradesh Rajarshi Tandon  
Open University

Bachelor in Computer  
Application

# BCA-118

## Windows Programming

<b>Block-1</b>	<b>INTRODUCTION TO WINDOWS PROGRAMMING</b>	<b>3-82</b>
UNIT-1	Windows Programming	7
UNIT-2	Programming Resources	35
UNIT-3	Visual C++ Programming	53
<b>Block-2</b>	<b>VISUAL BASIC PROGRAMMING</b>	<b>83-172</b>
UNIT-4	Windows Programming	87
UNIT-5	Working with Controls	119
UNIT-6	Dialog Boxes and Internet	151
<b>Block-3</b>	<b>WORKING WITH GRAPHICS</b>	<b>173-230</b>
UNIT-7	Document View Architecture	177
UNIT-8	Graphics and Multimedia	203
<b>Block-4</b>	<b>INTERFACING AND DATABASE APPLICATION</b>	<b>231-352</b>
UNIT-9	Interfacing Other Applications	235
UNIT-10	Database Application	259
UNIT-11	Network Programming	293
UNIT-12	Advanced Topics and Case Study	325





॥ सरस्वती नः सुभगा मयस्कृत ॥

Uttar Pradesh Rajarshi Tandon  
Open University

Bachelor in Computer  
Application

**BCA-118**

**Windows Programming**

**BLOCK**

**1**

**INTRODUCTION TO WINDOWS PROGRAMMING**

---

**UNIT-1**

**Windows Programming**

---

**UNIT-2**

**Programming Resources**

---

**UNIT-3**

**Visual C++ Programming**

---

---

## Course Design Committee

---

**Prof. Ashutosh Gupta**

Director

School of Science, UPRTOU Prayagraj

**Prof. Suneeta Agarwal**

Dept. of Computer Science & Engineering

Motilal Nehru National Institute of Technology, Allahabad, Prayagraj

**Dr. Upendra Nath Tripathi**

Associate Professor

Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

**Dr. Ashish Khare**

Associate Professor

Dept. of Computer Science, University of Allahabad, Prayagraj

**Ms. Marisha**

Assistant Professor (Computer Science)

School of Science, UPRTOU Prayagraj

**Mr. Manoj Kumar Balwant**

Assistant Professor (Computer Science)

School of Sciences, UPRTOU Prayagraj

---

## Course Preparation Committee

---

**Dr. Krishan Kumar**

**Author**

Assistant Professor

Department of Computer Science,

Gurukula Kangri Vishwavidyalaya Haridwar (UK)

**Dr. Brajesh Kumar**

**Editor**

Associate Professor, Dept. of CS & IT

M.J.P Rohilkahand University, Bareilly, Uttar Pradesh

**Prof. Ashutosh Gupta**

**Director (In-Charge)**

School of Computer & Information Sciences

UPRTOU Prayagraj

**Mr. Manoj Kumar Balwant**

**Coordinator**

Assistant Professor (computer science)

School of Sciences, UPRTOU Prayagraj

---

©UPRTOU, Prayagraj - 2020

ISBN :

---

©All Rights are reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the **Uttar Pradesh Rajarshi Tondon Open University, Prayagraj.**

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.

**Printed By :** Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

---

## BLOCK INTRODUCTION

---

We believe effective programmers must combine theory with practice, so they can adapt to ever-changing computing environments. This block does not cover the breadth of topics found in some professional reference books, but it has a number of features that make it useful in the classroom:

- A step-by-step learning approach in which new ideas and concepts build on existing ones.
- Check-up exercises at the end of each section.
- Review questions and programming exercises at the end of each chapter.

*Block 1* basically contains three units intended with Windows programming and its associated concepts like traditional programming, programming resources, and visual C++ programming.

*Unit 1* introduces the history, evolution and notion of Windows programming and also describes its fundamental principles. Basically, Windows programming is a programming language which intends with the graphical user interface (GUI). It gives the overview of traditional programming paradigms like procedure oriented programming. Moreover, event driven programming like Visual Basic and Visual C++ has also been discussed. After that windows messages' functions have been discussed. Lastly data link libraries (DLL) and SDK have been explained briefly in simple way.

*Unit 2* covers the basic need of programming resources like accelerators, bitmaps, dialog boxes, icons, menus, string tables, toolbars, and versions. These resources play an important role in windows programming development environment. These topics have been one-by-one explained using appropriate syntax and examples.

*Unit 3* gives the detailed idea of event driven programming in VC++. Event-driven, meaning code remains idle until called upon to respond to some event (button pressing, mouse up, mouse down, key press, menu selection etc). Conceptually, VC++ is governed by an event processor. Nothing happens until an event is detected. Once an event is detected, the code corresponding to that event (event procedure) is executed. Unit starts from data types and ends with MFC (Microsoft Foundation Classes) file handling. VC++ is the language with added GUI features originated from C++. As C++ is a partly object oriented programming language, VC++ explores all the characteristics of object oriented language like encapsulation, inheritance, polymorphism etc. MFCs are new features in VC++ which were not available in C++.



---

# UNIT 1 WINDOWS PROGRAMMING

---

## Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Introduction to Windows Programming
- 1.3 Traditional Programming Paradigms
- 1.4 Event Driven Programming
- 1.5 Handles and Data Types
- 1.6 Windows Messages
- 1.7 Device Contexts
- 1.8 Document Interfaces
- 1.9 Document Linking Libraries
- 1.10 Software Development Kit Tools
- 1.11 Context Help
- 1.12 Summary
- 1.13 Terminal Questions

---

## 1.0 INTRODUCTION

---

Windows programs are usually different from the conventional programs. The most obvious difference is to work with the graphical user interface (GUI) between user and program. Windows programs use a GUI, where in addition to use the keyboard; the user can do mouse clicks on buttons in GUI or select items from lists or drop-down menus. As, all Windows programs use the same interface, therefore provide a familiar feel to the user. More basically however, the Windows programs also operate in a completely different way from conventional programs. A conventional program runs without a break (apart from occasional pauses to request user input) upon execution, performing its tasks in a set order determined by the program logic. When it has finished its tasks, it terminates automatically. A typical Windows program works differently. When it is first executed, a new window opens on the desktop but then the program stops running and just waits until the user chooses to do something. Tasks are performed at the user's request when a menu item is selected or a button is clicked. It is the user, therefore, rather than the program who decides the order in which tasks are performed. After the completion of a task, the program goes back to waiting, and it does not terminate automatically but waits until the user requests it to do so.

Windows programs normally spend most of the time in waiting and doing nothing. While one program is waiting, other Windows programs may be executing. The operating system controls execution of programs sharing the CPU time between all the programs that are running. Windows programs are event-driven programs. An event is a user action such as the button click, the press of a key on the keyboard, or the selection of an item from a drop-down menu or list. The Windows program responds to each event by taking an action and the program code that responds to the event is called an event handler, typically, a subroutine or function. There must be an event handler for every possible Windows event.

This unit deals with the basic concept of traditional and Windows programming. Readers might be familiar with the conventional programming languages like C, C++, Java etc. Windows programming is slightly different as discussed earlier. Before knowing about the Windows programming, understanding of operating system (OS) is necessary. There are many well-known operating systems like DOS, Windows, Linux, Android, and Macintosh. The operating system is an interface between the user and hardware. It acts as the manager which manages all the resources of computer like processor, memory; hardware devices such as printer, monitor, scanner, and modems. Moreover, an operating system is termed as the soul of the machine without which it is too difficult to operate the machine. Any application cannot directly interact with the hardware or resources rather it needs a medium or interface. Therefore, all the applications interact with the operating system to obtain the access to the available resources. There are many operating systems based on the specific requirements such as single-user operating system, multi-user operating system, network operating system, distributed operating system, and handheld operating system etc. DOS and Windows are two operating systems popular in the world of Information Technology. Windows is a user friendly operating system and widely popular.

The Windows programming is closely related to the Windows operating system. Microsoft released the first version Windows 1.0 of Windows on November 20, 1985. It was a multi-application operating system i.e. able to interact with multiple applications simultaneously. In windows operating system, hardware is shared by all users/applications. It changed from 16-bit to 32-bit architecture when Windows NT and Windows 95 were introduced later. Windows 10 is the latest version of Windows OS for desktops.

The windows API or WinAPI is Microsoft's core set of application programming interfaces which exist in operating systems. Windows API are also known as Win32 API. The windows API is mainly focussed on the programming language where the data structures and functions are defined. API may also be used by any language compiler or assembler to handle the well-defined low level data structures. The internal implementation of APIs has been developed using several languages. Despite the fact that C language is not the object oriented programming language however, Windows API and Windows both have been considered as object oriented. Wrapper classes and MFC make APIs more explicit. Earlier there were only two ways to access the Windows API, one was to use the C programming (mostly used) and other was Microsoft Pascal programming (rarely used).



---

## 1.1 OBJECTIVES

---

At the end of this unit you will come to know about the following:

- Definition of windows programming
- The mechanism of Windows programming
- Traditional programming
- Data Types
- Windows Messages
- Device Contexts
- Document Interfaces
- Dynamic Linking Libraries
- Context Help

---

## 1.2 INTRODUCTION TO WINDOWS PROGRAMMING

---

Prior to Windows GUI, DOS was very popular and widely used in computers. DOS is a single user operating system. On the other hand, with the added features, Windows, developed by Microsoft grew dramatically. As Windows was a 16-bit graphical layer for MS-DOS originally; it grew and gained the ability to handle 32-bit programs and eventually became totally 32-bit when Windows NT (in 1993) and Windows 2000 came out. After Windows 95 in 1995, Microsoft began to remove dependencies on DOS and finally fully implemented the separation in Windows 2000. Moreover, Windows has many advanced features as well as many platform specific issues. It possesses an API that consists of thousands of mostly undocumented GUI functions as well as having varying degrees of MS-DOS compatibility. Furthermore, with the advent of NT (New Technology), Windows relied completely on the NT kernel instead of its MS-DOS subsystem. The NT kernel is capable of emulating the necessary DOS functionality. In addition to the NT kernel, Microsoft had also introduced many API wrappers, such as the Microsoft Foundation Classes (MFCs), Component Object Model (COM), and .NET technologies. Over the years, the most popular languages for use on Windows include Visual Basic/VB6.0 and C/C++; although C++ is quickly being replaced by the .NET platform, specifically by C# (C Sharp). So It can be said that VB and C# are the two most commonly used languages to develop Windows GUI.

Windows 1.0, 2.0, and 3.11 are considered to be an older generation of Windows systems that were built to be a simple graphical layer over the MS-DOS operating system. Windows 95, Windows 98, and Windows ME were designed to bypass MS-DOS (although DOS was still present), and were all based on the same code structure known as the "9x Kernel". Windows NT 4.0, Windows 2000,

Windows XP, Windows Vista, Windows 7, Window 8, and Windows Server are all based on a collection of code known as the "NT Kernel".

The Windows system might be surprising for some people to learn, it is a very hands-on system. This is not a familiar concept for people who are just beginning C programming using the standard library. In a normal software project, there is typically a main function, and the main function in turn calls other functions that are defined in a project. In a Windows function, typically the programmer provides function pointers to the system, and Windows makes calls for them into the program. Also, in a Windows program, the code will sit idle when there is nothing to be done. Using the *message loop architecture*, Windows will send messages to the program when an event needs to be handled, and the program responds to the messages. If the program doesn't respond, the message is ignored.

The new Windows Store applications represent a radical break with traditional Windows. The programs generally run in a full-screen mode—although two programs can share the screen in a “snap” mode—and many of these programs will probably be optimized for touch and tablet use. These applications are purchasable and installable only from the application store run by Microsoft. As a developer, one can deploy and test applications directly from Visual Studio. In addition to the versions of Windows that run on x86 processors, there is also a version of Windows that runs on *ARM processors*, most commonly found in low-cost tablets and other mobile devices. This version of Windows is called Windows RT, and it only comes pre-installed on these machines. One of the first computers running Windows RT is the initial release of the Microsoft Surface. Aside from some preinstalled desktop applications, Windows RT runs new Windows Store applications only.

**Note:** - *An ARM processor is one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM)*

Windows is a GUI operating system, developed by Microsoft, has several versions and each version has different GUI with a good desktop display that allows users to view files and folders. It is widely used operating system for personal computers. Microsoft Windows is designed for home computing and professional computing. The versions of Windows home editions are:

- Windows 1.0 (1985)
- Windows 2.0 (1987)
- Windows 3.0 (1990)
- Windows 3.1 (1992)
- Windows NT (1993)
- Windows 95 (1995)
- Windows 98 (1998)
- Windows 98SE (1999)

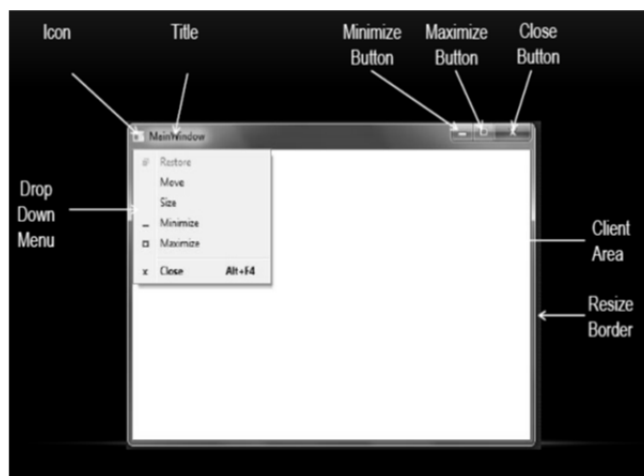
- Windows Me (2000)
- Windows XP (2001)
- Windows Vista (1990)
- Windows 7 (2008)
- Windows 8 (2010)
- Windows 10 (2013)

Windows programming languages and their development environments are different in different years as given in Table 1.1.

**Table1.1 Windows Programming Languages and their Platforms**

Year	Language	Platform
1985	C or VB	Windows Application Program Interface (API)
1992	C++ or VB	Microsoft Foundation Class (MFC) library
2001	C# or C++	Windows Forms (.NET Framework) or VB
2006	C# or VB	Windows Presentation Foundation (WPF)

Two main things in Windows are System 32 and Win API. System32 directory is located in *C:\Windows\System32*. In addition, Windows development platforms are: Win16API, Win32API, Window foundations classes, .NET Window forms (WinForms), .NET Window Presentation Foundation (WPF). On the other hand, Win API provides the environment to develop desktop and server applications. Similar functions are provided by 32-bit and 64-bit operating systems. Windows applications can be developed using a procedure-oriented approach in C or C++. Windows provides the ability to develop GUI.



**Fig. 1.1 Windows Structure**

Windows is a graphic-based multi-tasking operating system. All the programs have a consistent look and command structure. For the development of Windows applications, it provides various predefined functions that allow easy implementations of menus, dialog boxes, scroll bars, icons to represent a better user-friendly interface. It also provides a hardware-independent environment to the programs.

To write a Windows program in C or C++ you need to install a Microsoft Windows Software Development Kit (SDK) or an environment which includes SDK such as Visual C++. Usually SDK contains the headers and libraries necessary to compile and link an application. Windows SDK also includes command-line tools for building Windows application, including the Visual C++ compiler and linker. Although you can compile and run the programs at command-line, however full-featured development environment like Microsoft Visual C++ is recommended.

Moreover, as Windows is a user friendly operating system which provides many features, it has MFC, a C++ class library, which provides an object oriented environment around the Windows API. Object oriented concepts like abstraction, inheritance, polymorphism, and data hiding etc. provides characteristics like robustness, flexibility, dynamicity, privacy, reusability etc. to Windows programming. The main magic of Windows is its MFC which contains nearly more than 200 predefined classes.

Furthermore, MFC is also an *application framework*. More than merely a collection of classes, MFC helps define the structure of an application and handles many routine chores on the application's behalf. Starting with *CWinApp*, the class that represents the application itself, MFC encapsulates virtually every aspect of a program's operation. The framework supplies the *WinMain* function, and *WinMain* in turn calls the application object's member functions to make the program go. One of the *CWinApp* member functions called by *WinMain*—*Run*—provides the message loop that pumps messages to the application's window. The framework also provides abstractions that go above and beyond what the Windows API has to offer. For example, MFC's document/view architecture builds a powerful infrastructure on top of the API that separates a program's data from graphical representations, or views, of that data. Such abstractions are totally foreign to the API and don't exist outside the framework of MFC or a similar class library.

---

## 1.3 TRADITIONAL PROGRAMMING PARADIGMS

---

Traditional programming like C, C++, and Java etc. are used for the most of the applications as well as system software. Programs written for traditional operating environment use a procedural programming model in which programs execute from top to bottom in an orderly fashion. The path taken from start to finish may vary with each invocation of the program depending on the input it receives or the conditions under which it is run, but the path remains fairly predictable. In a C program, execution begins with the first line in the function named *main* and ends when *main* returns. In between, *main* might call other functions and these functions might call even more functions, but ultimately it is

the program—not the operating system—that determines what gets called and when.

Windows programs operate differently. They use the event-driven programming model, in which applications respond to *events* by processing messages sent by the operating system. An event could be a keystroke, a mouse-click, or a command for a window to repaint itself, among other things. The entry point for a Windows program is a function named *WinMain*, but most of the action takes place in a function known as the *window procedure*. The window procedure processes messages sent to the window. *WinMain* creates that window and then enters a *message loop*, alternately retrieving messages and dispatching them to the window procedure. Messages wait in a message queue until they are retrieved. A typical Windows application performs the bulk of its processing in response to the messages it receives, and in between messages, it does little except wait for the next message to arrive.

A few years ago, the person learning to program Microsoft Windows for the first time had a limited number of programming tools to choose from. C was the language spoken by the Windows Software Development Kit (SDK), and alternative Windows programming environments such as Microsoft Visual Basic hadn't arrived on the scene. Most Windows applications were written in C, and the fledgling Windows programmer faced the daunting task not only of learning the ins and outs of a new operating system but also of getting acquainted with the hundreds of different API functions that Windows supports.

Today many Windows programs are still written in C. But the variety of Windows programming environments available means that commercial-quality Windows programs can be written in C, C++, VC++, and a number of other languages. Moreover, C++ has all but replaced C as the professional Windows programmer's language of choice because the complexity of Windows, coupled with the wide-ranging scope of the Windows API, cries out for an object-oriented programming language. Many Windows programmers have concluded that C++ offers a compelling alternative to C that, combined with a class library that abstracts the API and encapsulates the basic behavior of windows and other objects in reusable classes, makes Windows programming simpler. And an overwhelming majority of C++ programmers have settled on the *Microsoft Foundation Class (MFC)* library as their class library of choice. Other Windows class libraries are available, but only MFC was written by the company that writes the operating system. MFC is continually updated to incorporate the latest changes to Windows itself, and it provides a comprehensive set of classes representing everything from windows to *ActiveX controls* in order to make the job of writing Windows applications easier.

If one is coming to MFC from a traditional Windows programming environment such as C and the Windows SDK already familiar with many of the concepts need to understand Windows programming with MFC. But if coming from a character-oriented environment such as MS-DOS or UNIX, one finds that Windows programming is fundamentally different. This unit begins with an overview of the Windows programming model and under at how Windows applications work.

## CHECK YOUR PROGRESS

- Define main difference between traditional and windows programming.
- Write the names of all versions of Windows home editions.
- Give the names of programming languages used to develop Windows programs.

---

## 1.4 EVENT-DRIVEN PROGRAMMING

---

In computer science, event-driven programming is basically a programming method where the flow of control is determined by the events. These events are user actions e.g. keyboard actions, mouse actions, sensor inputs line in games, or some messages from other processes. Event-driven paradigm is an important paradigm which is the key part of GUI. This is similar to the device drivers e.g. USB device drivers which become active whenever a flash drive is inserted in a USB port. While, in embedded system the same thing happens by hardware interrupts. In all these situations, a main loop listens for events according to different actions performed. During every event, some kind of event handler works. Event handlers may be a trivial event handler (works with key, mouse, sensory input) or an exception event handler (works with hardware error, interrupts, underflow, and overflow). The languages which provide high-level-abstractions are used to develop even-driven programs.

---

### 1.4.1 BACKGROUND

---

Event driven programming entirely depends upon the events. Events are nothing but a moment in computer hardware when something instantaneously happens due to the interaction with the devices. For example, mouse-click, mouse-up, mouse-down, key-press, key-up, key-down, pressing a key, etc. all are the events. Earlier, ahead the advent of object oriented programming languages like C++, C#, and Java; events were implemented as subroutines within a procedural program. The flow of the such program is entirely decided by the programmer and controlled from within the main routine of the program. And the programs were highly structured having high complexity in the logic. The entire code of the program is written by the programmer, including the code for event and exception handling as well as the code required to manage the flow of program execution.

Usually, in a modern event-driven program, flow control is not recognized. The main routine is an *event-loop* that waits for an event to occur, and then invokes the appropriate event-handling routine. Since the code for this event loop is usually provided by the event-driven development environment or framework, and largely invisible to the programmer, the programmer's perception of the application is that of a collection of event handling routines. Programmers working with procedural programming languages, sometimes find that the

transition to an event-driven environment requires a considerable mental adjustment.

GUI changed the world of computer science completely. The change from procedural to event-driven programming has been accelerated by the introduction of the GUI which has been widely adopted for use in operating systems and end-user applications. It really began, however, with the introduction of *object-oriented* programming languages and development methodologies in the late 1970s. By the 1990's, object-oriented technologies had largely replaced the procedural programming languages and structured development methods. One of the drivers behind the object oriented approach to programming that emerged during that era was the speed with which database technology developed and was adopted for commercial use. Information system designers increasingly saw the database itself, rather than the software that was used to access it, as the central component of a computerized information system. The software simply provided a user interface to the database, and a set of event handling procedures to deal with database queries and updates.

---

## 1.4.2 INTRODUCTION

---

*Event-driven programming* is a programming paradigm in which the flow of program execution is always determined by *events*, for example a user action such as a mouse click, mouse up, mouse down, key press, key up or a message from the operating system or another program. An event-driven application is designed to detect events as they occur, and then deal with them using an appropriate *event-handling procedure*. The idea is an extension of *interrupt-driven programming* found in early command-line environments such as DOS, and in embedded systems (where the application is implemented as firmware). Event-driven programs can be written in any programming language, although some languages (Visual Basic for example) are specifically designed to facilitate event-driven programming, and provide an *integrated development environment* (IDE) that partially automates the production of code, and provides a comprehensive selection of built-in objects and controls, each of which can respond to a range of events. Virtually all object-oriented and visual languages support event-driven programming. Visual Basic, Visual C++ and Java are examples of such languages.

A visual programming IDE such as VB.Net provides much of the code for detecting events automatically when a new application is created. The programmer can therefore, concentrate on issues such as interface design, which involves adding controls such as command buttons, text boxes, and labels to standard *forms* (a form represents an application's workspace or *window*). Once the user interface is complete, the programmer can add event-handling code to each control as required. Many visual programming environments will even provide code templates for event-handlers, so the programmer only needs to provide the code that defines the action the program should take when the event occurs. Each event-handler is usually bound to a specific object or control on a form. Any additional subroutines, methods, or function procedures required are usually placed in a separate code module, and can be called from other parts of the program as and when needed.

---

### 1.4.3 HOW EVENT-DRIVEN PROGRAMMING WORKS

---

The main component of an event-driven application is a scheduler that receives a stream of events and passes each event to the relevant event-handler. The scheduler continues to remain active until it encounters an event that results it to terminate the application. Under certain circumstances, the scheduler may encounter an event for which it cannot assign an appropriate event handler. Depending upon the nature of the event, the scheduler can either ignore it or raise an error. Within an event-driven programming environment, standard events are usually identified using the ID of the object affected by the event (e.g. the name of a command button on a form), and the event ID (e.g. "left-click"). The information passed to the event-handler may include additional information, such as the  $x$  and  $y$  coordinates of the mouse pointer at the time the event occurred, or the state of the *Shift* key (if the event in question is a key-press).

Events are often actions performed by the user during the execution of a program, but can also be messages generated by the operating system or another application, or an interrupt generated by a peripheral device or system hardware. If the user clicks on a button with the mouse or hits the *Enter* key, it generates an event. If a file download completes, it generates an event. And if there is a hardware or software error, it generates an event. The events are dealt with by a central event-handler (usually called a *dispatcher* or *scheduler*) that runs continuously in the background and waits for an event to occur. When an event *does* occur, the scheduler must determine the type of event and call the appropriate event-handler to deal with it. The information passed to the event handler by the scheduler varies, but will include sufficient information to allow the event-handler to take any action necessary.

---

## 1.5 HANDLES AND DATA TYPES

---

In Windows, and generally in computing, a *handle* is an abstraction which hides a real memory address from the API user, allowing the system to reorganize physical memory transparently to the program. Resolving a handle into a pointer locks the memory, and releasing the handle invalidates the pointer. In this case think of it as an index into a table of pointers. You use the index for the system API calls, and the system can change the pointer in the table at will. Alternatively, a real pointer may be given as the handle when the API writer intends that the user of the API be insulated from the specifics of what the address returned points to. In this case, it must be considered that what the handle points to may change at any time (from API version to version or even from call to call of the API that returns the handle). The handle should therefore be treated as simply an opaque value meaningful only to the API. Moreover; In any modern operating system, even the so-called "real pointers" are still opaque handles into the virtual memory space of the process, which enables the OS to manage and rearrange memory without invalidating the pointers within the process.

Variables and constants are the basic data objects manipulated in a program. Declarations list the variables to be used stating the type of values they can store and sometimes also provide their initial values. Operators specify what



to be done with the data values provided. Expressions combine variables and constants to produce new values. The type of an object determines the set of values it can have and what operations can be performed on it. The programming standard has made many small changes and additions to basic types and expressions in programming languages. Characteristics of data types may be different for different programming languages.

---

## 1.5.1 HUNGARIAN NOTATION

---

Hungarian notation is a naming convention in computer [programming](#) that indicates either the type of [object](#) or the way it should be used. It was originally proposed by [Charles Simonyi](#), a programmer at [Xerox PARC](#) in the early 1980s. There are two variations of Hungarian notation: Systems and Apps. They both involve using a special [prefix](#) as part of the name to indicate an object's nature.

---

### 1.5.1.1 SYSTEMS HUNGARIAN NOTATION

---

In *Systems Hungarian notation*, the prefix represents the actual data type of the object. For instance, if the object named *Greeting* were a zero-terminated string, its Systems Hungarian name might be *szGreeting*. Or, if the object *YesOrNo* were a boolean variable, its Systems Hungarian name would be *bYesOrNo*.

---

### 1.5.1.2 APPS HUNGARIAN NOTATION

---

In *Apps Hungarian notation*, the prefix represents the logical data type, which gives an indication of the object's purpose. For instance, an "unsafe" (unsanitized) string might have the prefix *us*, and a variable used for counting might be prefixed with *n*.

The Win32 API uses the *Hungarian Notation* for naming variables. Hungarian Notation requires that a variable be prefixed with an abbreviation of its data type, so that when you are reading the code, you know exactly what type of variable it is. The reason behind this practice in the Win32 API is the availability of different data types makes it difficult to keep them all straight. Also, there are a number of different data types that are essentially defined the same way, and therefore some compilers do not pick up errors when they are used incorrectly. As we discuss each data type, we also note the common prefixes for that data type. In which each variable name begins with one or more lowercase characters identifying the variable's type: *h* for handle, *n* for integer, and so on. Prefixes are often combined to form other prefixes, as when *p* and *sz* are joined to form *psz*, which stands for "pointer to zero-terminated string."

Putting the letter "P" in front of a data type, or "p" in front of a variable usually indicates that the variable is a pointer. The letters "LP" or the prefix "lp" stands for "Long Pointer", which is exactly the same as a regular pointer on 32-bit machines. LP data objects are simply legacy objects that were carried over from Windows 3.1 or earlier, when pointers and long pointers needed to be differentiated. In modern 32-bit systems, these prefixes can be used interchangeably.

---

## 1.5.2 LPVOID

---

LPVOID data types are defined as being a "pointer to a void object". It seems to be strange to some programmers, but the ANSI-C standard allows for generic pointers to be defined as "void\*" types. This means that LPVOID pointers can be used to point to different types of objects, without creating a compiler error. However, the burden is on the programmer to keep track of what type of object is being pointed to. Also, some Win32 API functions may have arguments labeled as "LPVOID lpReserved". These reserved data members should not be used in the program, because they either depend on functionality that hasn't yet been implemented by Microsoft, or else they are only used in certain applications. If you see a function with an "LPVOID lpReserved" argument, you must always pass a NULL value for that parameter. Some functions will fail if you do not do so. LPVOID objects frequently do not have prefixes, although it is relatively common to prefix an LPVOID variable with the letter "p", as it is a pointer.

---

## 1.5.3 DWORD, WORD, BYTE

---

These data types are defined to be a specific length, regardless of the platform. There is a complexity in the header files to achieve this, but code becomes very well standardized, and very portable to different hardware platforms and different compilers.

**DWORDS (Double WORDs):** It is the most commonly used of these three data types. DWORDs are defined always to be unsigned 32-bit quantities. For any machine irrespective of it being 16-bit, 32-bit, or 64-bit, a DWORD is always 32 bits long. Hence, DWORDs are very common and popular on 32-bit machines, but less common on 16-bit and 64-bit machines.

**WORDs (Single WORDs):** These are defined strictly as unsigned 16-bit values, regardless of what machine you are programming on.

**BYTE:** These are defined strictly as being unsigned 8-bit values.

**QWORDS (Quad WORDs):** Although rare, these are defined as being unsigned 64-bit quantities.

Putting a "P" in front of any of these identifiers indicates that the variable is a pointer. Putting two "Ps" in front indicates it's a pointer to a pointer. These variables may be unprefixed, or they may use any of the prefixes common with DWORDs. Because of the differences in compilers, the definition of these data types may be different, but typically following definitions are used:

```
#include <stdint.h>

typedef uint8_t BYTE;

typedef uint16_t WORD;

typedef uint32_t DWORD;

typedef uint64_t QWORD;
```

Usually, we can define pointers to these types as:

```
#include <stdint.h>
typedef uint8_t * PBYTE;
typedef uint16_t * PWORD;
typedef uint32_t * PDWORD;
typedef uint64_t * PQWORD;
```

DWORD variables are typically prefixed with "dw". Likewise, we have prefixes given in the following Table 1.2.

**Table1.2 Data types & their prefixes**

Data Type	Prefix
BYTE	"b"
WORD	"w"
DWORD	"dw"
QWORD	"qw"

---

### 1.5.4 LONG, INT, SHORT, CHAR

---

These types are not defined to a specific length. It is left to the host machine to determine exactly how many bits each of these types take.

**LONG notation:** LONG variables are typically prefixed with "l" (lower-case L).

**UINT notation:** UINT variables are typically prefixed with an "i" or an "ui" to indicate that it is an integer or unsigned integer.

**CHAR, UCHAR notation:** These variables are usually prefixed with a "c" or an "uc" respectively.

If the size of the variable doesn't matter, you can use integer types. However, if you are concerned with the size of the variable, other data types such as BYTE, WORD, DWORD, or QWORD identifiers can be used. The sizes of these data types are platform-independent and never change. Syntax is given as follows:

```
typedef long LONG;
typedef unsigned long ULONG;
typedef int INT;
typedef unsigned int UINT;
```

```
typedef short SHORT;
typedef unsigned short USHORT;
typedef char CHAR;
typedef unsigned char UCHAR;
```

---

### 1.5.5 STR, LPSTR

---

**STR:** It is a string data type with storage already allocated. STR data types are used when the string is supposed to be treated as an immediate array, and not as a simple character pointer. The variable name prefix for a STR data type is "sz" because it is a zero-terminated string (ends with a null character). Most programmers however do not prefer STR opting instead a character array. Defining a string as an array allows the size to be set explicitly. Also, creating a large string on the stack can cause undesirable stack-overflow problems.

**LPSTR:** It stands for "Long Pointer to a STR" and it is more commonly used than STR. It is essentially defined as follows:

```
#define STR * LPSTR;
```

LPSTR can be used exactly like other string objects, except that LPSTR is explicitly defined as being ASCII, not unicode, and this definition will hold on all platforms. LPSTR variables would usually be prefixed with the letters "lpsz" to denote a "Long Pointer to a String that is Zero-terminated". The "sz" part of the prefix is important, because some strings in the Windows world (especially when talking about the DDK1) are not zero-terminated. LPSTR variables prefixed with the "lpsz" can all be used seamlessly with the standard library <string.h> functions.

---

### 1.5.6 TCHAR

---

TCHAR data type is a generic character data type. TCHAR can hold either standard 1-byte ASCII characters, or wide 2-byte Unicode characters. Because this data type is defined by a macro, only character data should be used with this type. TCHAR is similar to the following:

```
#ifdef UNICODE
#define TCHAR WORD
#else
#define TCHAR BYTE
#endif
```

---

### 1.5.7 TSTR, LPTSTR

---

Strings of TCHARs are typically referred to as TSTR data types. They are defined as LPTSTR types such as:

```
#define TCHAR * LPTSTR
```

These strings can be either UNICODE or ASCII, depending on the status of the UNICODE macro. LPTSTR variables are long pointers to generic strings, and may contain either ASCII strings or Unicode strings, depending on the environment. LPTSTR data types are also prefixed with the letters "lpsz".

---

## 1.5.8 HANDLE

---

HANDLE is another data type in Windows programming. It is slightly different and hard to understand for the new programmers migrated from other languages. It is also one of the most important data objects in win32 programming. The details of the data objects are maintained by the kernel. Basically buttons, icons, mouse pointers, windows etc. have the entry in a table and each entry is assigned a unique identifier. This identifier is called HANDLE. HANDLES are defined in *<windows.h>*, but they are not similar to the integers and hence should not be used like integers. In other words, HANDLES can be stored but cannot be changed by the programmers. Moreover, they are prefixed with an "h". HANDLES are unsigned integers that Windows uses internally to keep track of objects in memory. Windows moves objects such as memory blocks to make room in the memory. If the object is moved in memory, the handles table is updated. Some HANDLES are described below.

**HWND:** HWND variables are "Handles to a Window". These are used to keep track of the various objects that appear on the screen. To communicate with a particular window, you need to have a copy of the window's handle.

### Syntax:

```
HWND hwnd; // used for main window
```

```
HWND hwndChild1, hwndChild2...// used for child window
```

```
HWND hDlg; // used for dialog boxes
```

**HINSTANCE:** HINSTANCE variables are handles to a program instance. Each program gets a single instance variable and it is important for the kernel to communicate with the program. It is usually a benefit to make this HINSTANCE variable a global value, so that all your functions can access it when needed.

### Syntax:

```
HINSTANCE hInstance;
```

**HMENU:** If the program has any type of menu such as drop-down then that will be associated with a HANDLE. To perform some operations like alter, display etc. you need to access HMENU handle.

---

## 1.6 WINDOWS MESSAGES

---

After creation of the window, it interacts with rest of the system by means of messages. Normally the system sends message to the window, and window sends messages back to the system. This process lasts from the creation of

message till it ends. In fact, most of the programs read messages and respond to them. Messages come in the form of MSG data type which uses a function *GetMessage()* to read the message from the message queue. After this, translate function is used to do the simple tasks like conversion to Unicode etc. Finally, the message is sent to the window to process using the *DispatchMessage()* function.

### Example

```
MSG msg;
BOOL bRet;
while((bRet = GetMessage( &msg, NULL, 0, 0 )) != 0)
{
if (bRet == -1)
{
// handle the error and possibly exit
}
else
{
TranslateMessage(&msg);
DispatchMessage(&msg);
}
}
return msg.wParam;
```

---

## 1.6.1 TYPES OF MESSAGES

---

**WM\_CREATE:** This is the first message. Window receives this message only once, when it is first created. This message is used to perform tasks that need to be handled in the beginning, such as initializing variables, allocating memory, or creating child windows (buttons and textboxes).

**WM\_PAINT:** This message indicates that it is the time for the program to redraw itself. If you don't draw anything, then the window will either be a boring white (or grey) background, or if the background was not erased, will keep whatever image is already shown on it (which looks unstable). The **WM\_PAINT** message is sent when the system or another application makes a request to paint a portion of an application's window. The message is sent when the **UpdateWindow** or **RedrawWindow** function is called, or by the **DispatchMessage** function when the application obtains a **WM\_PAINT** message by using the **GetMessage** or **PeekMessage** function.

**WM\_COMMAND:** This is a general message that indicates that the user has done something on your window. Either the user has clicked a button, or the user has selected a menu item, or the user has pressed a special "Accelerator" key sequence. The WPARAM and LPARAM fields of WM\_COMMAND will contain some descriptions on what happened, so you can find a way to react to

this. If you do not process the `WM_COMMAND` messages, the user will not be able to click any buttons, or select any menu items.

**WM\_CLOSE:** If user decides to close the window, the kernel sends the `WM_CLOSE` message. This is the final chance to preserve the window as necessary. If you don't want it closed completely, you should handle the `WM_CLOSE` message and ensure that it does not destroy the window. If the `WM_CLOSE` message is passed to the `DefWindowProc`, then the window will next receive the `WM_DESTROY` message.

**WM\_DESTROY:** The `WM_DESTROY` indicates that a given window is removed from the screen and will be unloaded from memory. Normally, your program can post the `WM_QUIT` message to exit the program by calling `PostQuitMessage()`.

### **CHECK YOUR PROGRESS**

- Give the meaning of event driven programming.
- Define handles in windows programming.
- What do you mean by window messages and its types?

---

## **1.7 DEVICE CONTEXTS**

---

A *device context* is a structure that defines a set of graphic objects and their associated attributes, and the graphic modes that affect the output. The graphic objects include a pen for line drawing, a brush for painting and filling, a bitmap for copying or scrolling parts of the screen, a palette for defining the set of available colors, a region for clipping and other operations, and a path for painting and drawing operations.

Device Context is an abstraction in windows to access any output hardware like graphic adapters or printers which displays text and graphical drawings. Windows OS returns a handle to the device context of the device. Device context is a structure in kernel mode of the OS, where it stores the attributes and other properties of the device. Hardware may vary in many aspects but device context is common for all for any particular device type. Hardware vendors supply device dependent device drivers or miniport drivers for their hardware which takes care of the lower layer access and manageability of the hardware.

The upper layer is Win32 APIs to access the hardware from application layer. Windows provides a set of APIs to access a type of device through the device context. Lower layer access and manageability may vary from hardware to hardware and from vendor to vendor but methods and steps via Windows API are same for accessing same type of devices. It creates a uniform access point for device hardware. In other words, a device context is just a place where drawing occurs. So if you have two different DC's, you're drawing in two different places. It is a kind of file handle. Device contexts can refer to real-estate on screen, or to bitmaps that just reside in memory, and probably other places too.

Compatible contexts have the same underlying pixel organization. The pixel organization refers to the number of bits per pixel, bytes per pixel, colour organization and so forth. Memory bitmap device contexts can have any organization you want, but your screen contexts are going to be related (eventually) to buffers on your graphics card, which will (depending on mode, etc) have a very specific pixel organization. Having compatible contexts leads to efficient image data transfer between them, because little or no translation of the data is required. At the other extreme, if you have a 256 colour palette 8-bit map and you try to build it on a screen that has 8 bits each of RGBA per pixel, then every last pixel will require significant messaging as it is copied and so copying incompatible bitmaps is very much slower. According to Win32 SDK documentation, at least *BitBlt()* and *StretchBlt()* convert the source colour format to match the destination format, so it can be done.

*SelectObject()* controls which resources are currently active within the device context. A context has a current pen, brush, font, and bitmap. These make a lot of the other GDI calls simpler by allowing you to specify fewer parameters. For instance, you don't have to specify the font when you use *TextOut()*, but if you want to change the font, that's where *SelectObject()* comes in. If you feed *SelectObject()* a handle to a font, the return value is a handle to the font that was in effect, and subsequent operations use the new font. Behavior is the same for the other kinds of resources, pens, brushes, etc. When you want to draw on a graphics output device such as the screen or printer, you must first obtain a handle to a device context. By giving your program this handle, Windows is giving you permission to use the device. You then include the handle as an argument to the GDI functions to identify the device on which you wish to draw.

The device context contains many "attributes" that determine how the GDI functions work on the device. These attributes allow GDI functions to have just a few arguments, such as starting coordinates. The GDI functions do not need arguments for everything else that Windows needs to display the object on the device. For example, when you call *TextOut()*, you need to specify in the function only the device context handle, the starting coordinates, the text, and the length of the text. You don't need to specify the font, the color of the text, the color of the background behind the text, or the intercharacter spacing. These are all attributes that are part of the device context. When you want to change one of these attributes, you call a function that does so. Subsequent *TextOut* calls to that device context use the new attribute.

---

### 1.7.1 GETTING A DEVICE CONTEXT HANDLE

---

Windows provides several methods for obtaining a device context handle. If you obtain a video display device context handle while processing a message, you should release it before exiting the window procedure. After you release the handle, it is no longer valid. For a printer device context handle, the rules are not as strict. The most common method for obtaining a device context handle and then releasing it involves using the *BeginPaint* and *EndPaint* calls when processing the WM\_PAINT message:

```
hdc = BeginPaint (hwnd, &ps);
```



*[other program lines]*

```
EndPaint (hwnd, &ps);
```

The variable *ps* is a structure of type `PAINSTRUCT`. The *hdc* field of this structure is the same handle to the device context that *BeginPaint* returns. The `PAINSTRUCT` structure also contains a `RECT` (rectangle) structure named *rcPaint* that defines a rectangle encompassing the invalid region of the window's client area. With the device context handle obtained from *BeginPaint* you can draw only within this region. The *BeginPaint* call also validates this region.

Windows programs can also obtain a handle to a device context while processing messages other than `WM_PAINT`:

```
hdc = GetDC (hwnd);
```

*[other program lines]*

```
ReleaseDC (hwnd, hdc);
```

This device context applies to the client area of the window whose handle is *hwnd*. The primary difference between the use of these calls and the use of the *BeginPaint* and *EndPaint* combination is that you can draw on your entire client area with the handle returned from *GetDC*. However, *GetDC* and *ReleaseDC* don't validate any possibly invalid regions of the client area.

A Windows program can also obtain a handle to a device context that applies to the entire window and not only to the window's client area:

```
hdc = GetWindowDC (hwnd);
```

*[other program lines]*

```
ReleaseDC (hwnd, hdc);
```

This device context includes the window title bar, menu, scroll bars, and frame in addition to the client area. Applications programs rarely use the *GetWindowDC* function. If you want to experiment with it, you should also trap the `WM_NCPAINT` ("nonclient paint") message, which is the message Windows used to draw on the nonclient areas of the window.

The *BeginPaint*, *GetDC*, and *GetWindowDC* calls obtain a device context associated with a particular window on the video display. A much more general function for obtaining a handle to a device context is *CreateDC*:

```
hdc = CreateDC (pszDriver, pszDevice, pszOutput, pData);
```

*[other program lines]*

```
DeleteDC (hdc);
```

For example, you can obtain a device context handle for the entire display by calling

```
hdc = CreateDC (TEXT ("DISPLAY"), NULL, NULL, NULL);
```

Writing outside your window is generally impolite, but it's convenient for

some unusual applications. (Although this fact is not documented, you can also retrieve a device context for the entire screen by calling *GetDC* with a NULL argument.)

Sometimes you need only to obtain some information about a device context and not do any drawing. In these cases, you can obtain a handle to an "information context" by using *CreateIC*. The arguments are the same as for the *CreateDC* function. For example,

```
hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL);
```

You can't write to the device by using this information context handle. When working with bitmaps, it can sometimes be useful to obtain a "memory device context":

```
hdcMem = CreateCompatibleDC (hdc);
```

```
[other program lines]
```

```
DeleteDC (hdcMem);
```

You can select a bitmap into the memory device context and use GDI functions to draw on the bitmap. A metafile is a collection of GDI function calls encoded in binary form. You can create a metafile by obtaining a metafile device context:

```
hdcMeta = CreateMetaFile (pszFilename);
```

```
[other program lines]
```

```
hmf = CloseMetaFile (hdcMeta);
```

During the time the metafile device context is valid, any GDI calls you make using *hdcMeta* are not displayed but become part of the metafile. When you call *CloseMetaFile*, the device context handle becomes invalid. The function returns a handle to the metafile (*hmf*).

---

## 1.7.2 GETTING DEVICE CONTEXT INFORMATION

---

A device context usually refers to a physical display device such as a video display or a printer. Often, you need to obtain information about this device, including the size of the display, in terms of both pixels and physical dimensions, and its color capabilities. You can get this information by calling the *GetDeviceCap* ("get device capabilities") function:

```
iValue = GetDeviceCaps (hdc, iIndex);
```

The *iIndex* argument is one of 29 identifiers defined in the WINGDI.H header file. For example, the *iIndex* value of *HORZRES* causes *GetDeviceCaps* to return the width of the device in pixels; a *VERTRES* argument returns the height of the device in pixels. If *hdc* is a handle to a screen device context, that's the same information you can get from *GetSystemMetrics*. If *hdc* is a handle to a printer device context, *GetDeviceCaps* returns the height and width of the printer display area in pixels.

You can also use *GetDeviceCaps* to determine the device's capabilities of processing various types of graphics. This is usually not important for dealing with the video display, but it becomes more important with working with printers. For example, most pen plotters can't draw bitmapped images and *GetDeviceCaps* can tell you that.

---

## 1.8 DOCUMENT INTERFACES

---

The document interface is the view, which can be provided to the user. **It joins** the things between the user and computer. Windows is intended to take the responsibility to provide interface in a very simple manner so that the end user could understand the environment easily in which one wants to work, and hence could do the different tasks like data submission, performing queries, retrieving the data, changing the data etc. Basically document interfaces are divided into two categories: Single-Document Interface (SDI) and Multiple-Document Interface (MDI).

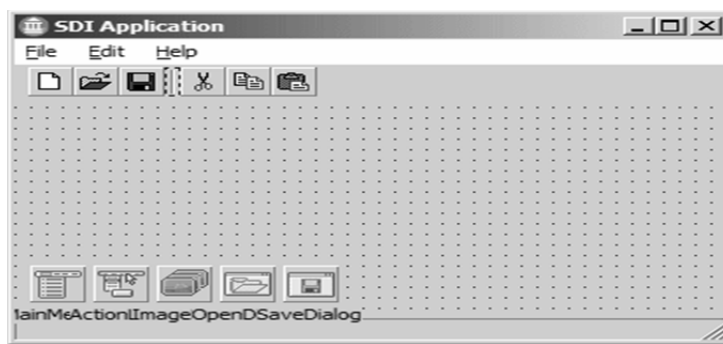
---

### 1.8.1 SINGLE DOCUMENT INTERFACE

---

In a Single Document Interface (SDI) application, there is only one window in each instance of the application. If you want a second window you can start a second complete instance of the application. Switching between windows happens at the operating system level when switching takes place between applications. In Microsoft Windows this means that you use the taskbar to select a different window, as all windows have an icon in the taskbar.

SDI is a type of GUI in which a program is unable to display more than one document in a window at time. This is the simplest interface which can be provided to the user. Basically it is a single-screen program in which data can be entered. SDI is simpler than multiple document interface as it is easy to program using SDI and is easy to use as well if implemented properly. However, it limits the multitasking capability of the program. And too many instances may create problem for the hardware. Due to this problem nowadays web browsers use the tabs. Examples of SDI are windows notepad, wordpad, calculator, command prompt, internet explorer 6 or earlier versions, etc. A window with SDI application is shown in Fig. 1.2.



**Fig. 1.2 SDI Application**

---

## 1.8.2 MULTIPLE DOCUMENT INTERFACE

---

The ‘classic’ Multiple Document Interface (MDI) design has been with us for many years now, and at one stage was a very common way of handling the user interface problems described so far. A MDI application has a main ‘shell’ MDI window with just a menu bar and possibly a toolbar. The user can load individual screens from these bars, and the screens will just ‘float’ within the window.

Typically, these applications have a ‘Window’ menu option, with ‘Tile’ and ‘Cascade’ options that rearrange all open windows. We also usually have a list of open windows on the ‘Window’ menu to allow us to find an individual screen. Microsoft uses this paradigm for Microsoft Word. If you open two documents in MS Word, you get two separate instances of the application.

---

## 1.9 DYNAMIC LINKING LIBRARIES

---

Basically dynamic link libraries or simply DLLs provide a mechanism for reused, shared and changed code and data, allowing a programmer to upgrade functionality without requiring applications to be re-linked or re-compiled. DLLs are the libraries which are helpful during the runtime without extra burden of memory. It was introduced with the first version of Microsoft Windows OS. And today it is the fundamental component of OS. They are not only the part of core OS but also the key component of many technologies like .NET and MFC.

DLLs allow a code fragment to be compiled into a single library and then linked by multiple programs. Which means only a single copy is needed and various programs share the data and functions. The main difference between a static and a dynamic library is that when a program is compiled it is not compiled rather it remains as a distinct module. This reduces the size of the memory and accessed or loaded into the memory whenever needed. The method of building a DLL file depends on the compiler used. However, the way of programming a DLL file is same.

---

### 1.9.1 /DLL hell

---

It is a common problem of the Windows programmers known as “DLL hell” and it doesn’t really have a solution. The term hell was coined in 90s. This is basically related with the permissions given by OS to let incorrect versions to be loaded upon the request of application that may result to a system crash.

---

### 1.9.2 \_\_DECLSPEC

---

This is a keyword, which is not for ANSI C standard, though most compilers understand it. Specifically, there are two `__declspec` identifiers:

`__declspec(dllexport)`

`__declspec(dllimport)`

When writing a DLL, we need to use the *dllexport* keyword to denote functions that are going to be available to other programs. Functions without this keyword will only be available for use from inside the library itself. Here is an example:

```
__declspec(dllexport) int MyFunc1(int foo)
```

The *\_\_declspec* identifier for a function needs to be specified both in the function prototype and the function declaration, when building a DLL. To "import" a DLL function into a regular program, the program must link to the DLL, and must prototype the function to be imported using the *dllimport* keyword such as:

```
__declspec(dllimport) int MyFunc1(int foo);
```

Now the program can use the function, even though the function exists in an external library. The compiler works with Windows to handle all the details. Some people find it useful to define a single header file for their DLL instead of maintaining one header file for building a DLL. Following is the macro that is common in DLL creation:

```
#ifdef BUILDING_DLL
#define DLL_FUNCTION __declspec(dllexport)
#else
#define DLL_FUNCTION __declspec(dllimport)
#endif
```

Now, to build the DLL, we need to define the *BUILDING\_DLL* macro. When we import the DLL, we don't need to use that macro. Functions then can be prototyped are:

```
DLL_FUNCTION int MyFunc1(void);
DLL_FUNCTION int MyFunc2(void);
.....
```

---

### 1.9.3 DLLMAIN

---

When the Windows links to a DLL program it calls Dll main function. This means that every Dll needs a main function. Dll main can be defined as:

```
BOOL WINAPI DllMain (HINSTANCE hInstance, DWORD reason, LPVOID reserved)
```

The keywords "BOOL", "APIENTRY", "HINSTANCE", etc. all are defined in <windows.h>. So, you must include this file even if you don't use any Win32 API functions in your library. *APIENTRY* is just a keyword that the Windows uses internally. So, you don't need to worry about it. The variable "hInstance" is the HINSTANCE handle for the library. You can keep, use, or trash it depending on one of the following four different values:

**DLL\_PROCESS\_ATTACH:** a new program has just linked to the library for the first time.

**DLL\_PROCESS\_DETACH:** a program has unlinked the library.

**DLL\_THREAD\_ATTACH:** a thread from a program has linked to the library.

**DLL\_THREAD\_DETACH:** a thread from a program has just unlinked the library.

The DllMain function doesn't need to do anything special for these cases, although some libraries will find it useful to allocate storage for each new thread or process that is being used with the library. The DllMain function must return TRUE if the library loaded successfully, or FALSE if the library had an error and could not load. If you return FALSE, the program will pop up a warning message and crash.

---

## 1.9.4 DLL BENEFITS

---

**Software Engineering Perspective:** Increasing the reusability of common routines. Easy to upgrade by changing the libraries only.

**System Utilization Perspective:** The code segment can be sharing at runtime; decrease the consume of memory and disk space. When multiple programs use the same function library, a DLL can reduce the duplication of code that is loaded onto disk and physical memory. This can greatly influence the performance not only of the program that runs in the background, but also other programs running on the Windows operating system.

**They promote a modular structure:** A DLL helps you to promote programs that have a modular structure in development. This enables us to develop a large number of programs that require multiple language versions or a program that requires a modular structure. An example of a modular program is an accounting program that has many modules that can be loaded dynamically at the time of execution.

**Facilitate implementation and installation:** When a function within a DLL requires an update or a fix, it is not necessary that the program be downloaded again. Additionally, if multiple programs use the same DLL, all programs will benefit from update or remediation.

---

## 1.10 SOFTWARE DEVELOPMENT KIT TOOLS

---

A *software development kit (SDK)* is basically a set of software development tools that allows the creation of applications for a specific software package, software framework, hardware platform, desktop computer, laptop, video game console, operating system, or similar development platform. To enhance applications with advanced features, advertisements, push notifications, and more, most applications' developers implement certain SDKs. Some SDKs are critical for developing a platform-specific app. For example, the development of an Android app on Java platform requires a Java Development Kit (JDK); for iOS apps the iOS SDK; and for Universal Windows

Platform the .NET Framework SDK. Moreover, there are some SDKs that are installed in apps to provide analytics and data about application activity. The prominent creators of these types of SDKs include Google, InMobi, and Facebook.

An SDK can take the form of a simple implementation of one or more application programming interfaces (APIs) in the form of on-device libraries to interface to a particular programming language, or it may be as complex as hardware-specific tools that can communicate with a particular embedded system. Common tools include debugging facilities and other utilities, often presented in an integrated development environment (IDE). SDKs may also include sample code and technical notes or other supporting documentation such as tutorials to help clarify points made by the primary reference material.

SDKs often include licenses that make them unsuitable for building software intended to be developed under an incompatible license. For example, a proprietary SDK is generally incompatible with free software development, while a GPL-licensed SDK could be incompatible with proprietary software development, all particularly for legal reasons. However, SDKs built under Lesser General Public License (LGPL) are typically safe for proprietary development. The average Android mobile app implements on an average 15.6 separate SDKs, while gaming apps implementing on average 17.5 different SDKs. The most popular SDK categories for Android mobile apps are analytics and advertising.

SDKs can be unsafe because they are implemented within apps, but yet run separate code. Malicious SDKs (with honest intentions or not) can violate users' data privacy, damage app performance, or even cause apps to be banned from Google Play Store or the App Store. New technologies allow app developers to control and monitor client SDKs in real time. Providers of SDKs for specific systems or subsystems sometimes substitute a more specific term instead of the word “software”. For instance, both Microsoft and Citrix provide a driver development kit (DDK) for developing device drivers.

---

## 1.11 CONTEXT HELP

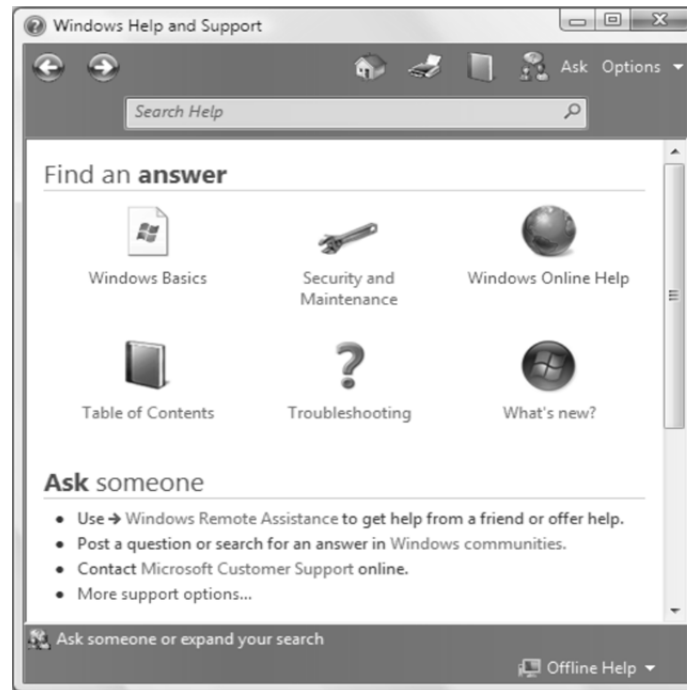
---

A Help system is composed of various types of contents designed to assist users when they are unable to complete a task, and want to understand a concept in more detail, or need more technical details than are available in the GUI. The GUI is the primary place because that is where users first try to solve their problems. They consult the Help system only if they can't accomplish their task with the GUI.

Context-help is a kind of help which is given basically online or called online help. It can be obtained from a specific state of the software which provides the help for the current situation. *Context-help* is also known as *Context-sensitive help*. Context-sensitive help, as opposed to general online help or online manuals, does not need to be accessible for reading as a whole. Each topic is supposed to describe extensively one state, situation, or feature of the software.

Context-sensitive help can be implemented using tooltips, which either provides a brief description of a GUI gadget or display a complete topic from the

help file. Other commonly used ways to access context-sensitive help start by clicking a button. One way uses a per widget (a small gadget) button that displays the help immediately. Another way changes the pointer shape to a question mark, and after the user clicks a widget, the help appears. Context-sensitive help examples include Apple's System 7 Balloon help, Microsoft's WinHelp, OS/2's INF Helper Sun's JavaHelp. A snapshot of Windows Help and Support is given in Fig. 1.3



**Fig. 1.3 Windows Help and Support**

A similar topic is *embedded help*, which can be thought of as a "deeper" context-sensitive help. It generally goes beyond basic explanations or manual clicks by either detecting a user's need for help or offering a guided explanation in situation. Embedded help is not to be confused with a software wizard.

### **CHECK YOUR PROGRESS**

- Give the meaning of device-context.
- Write briefly about SDK.
- What do you mean by context help?

---

## **1.12 SUMMARY**

---

In Windows, a *handle* is an abstraction which hides a real memory address from the API user, allowing the system to reorganize physical memory transparently to the program. Resolving a handle into a pointer locks the memory, and releasing the handle invalidates the pointer.



*Variables* and *constants* are the basic data objects manipulated in a program. Declarations list the variables to be used, and state what type they have and perhaps what their initial values are. Operators specify what is to be done to them. Expressions combine variables and constants to produce new values. The type of an object determines the set of values it can have and what operations can be performed on it. The programming standard has made many small changes and additions to basic types and expressions in programming languages. Characteristics of data types may be different for different programming languages.

After creation of the window, it interacts with rest of the system by means of *window messages*. Normally the system sends message to the window, and windows sends messages back to the system. This process lasts from the creation of message till it ends. In fact, most of the programs read messages and respond to them. Messages come in the form of MSG data type which uses a function *GetMessage()* to read the message from the message queue.

A *device context* is a structure that defines a set of graphic objects and their associated attributes, and the graphic modes that affect output. The graphic objects include a pen for line drawing, a brush for painting and filling, a bitmap for copying or scrolling parts of the screen, a palette for defining the set of available colors, a region for clipping and other operations, and a path for painting and drawing operations.

The *document interface* is the view which can be provided to the user. This is the joining thing between the user and computer. Basically document interfaces are divided into two categories: SDI and MDI. In a SDI application there is only one window in each instance of the application.

A MDI has a main 'shell' MDI window with just a menu bar and possibly a toolbar. The user can load individual screens from these bars, and the screens will just 'float' within the window.

DLLs provide a mechanism to reuse, share, and change code and data, allowing a programmer of code/data to upgrade functionality without requiring applications to be re-linked or re-compiled. DLLs are the libraries, which are helpful during the runtime without extra burden of memory. It was introduced with the first version of Microsoft Windows OS. And today it is the fundamental component of OS. They are not only the part of core OS but also the key component of many technologies like *.NET* and *MFC*.

SDK is basically a set of software development tools that allows the creation of applications for a specific software package, software framework, hardware platform, desktop, computer, laptop, video game console, operating system, or similar development platform. To enhance applications with advanced features, advertisements, push notifications, and more, most applications' developers implement certain software development kits.

Context-help is a kind of help which is given basically online or called online help. It can be obtained from a specific state of the software which provides the help for the current situation. Context-help is also known as Context-sensitive help.

---

## 1.13 TERMINAL QUESTIONS

---

1. What do you understand by Windows programming? Explain briefly.
2. Compare Windows programming with traditional programming.
3. Explain the importance of handles in Windows programming.
4. Write a short note on device context.
5. Write a short note on the role of DLLs in Windows programming.
6. Discuss the various types of messages.
7. Discuss about Context-help.
8. Write the significance of Window messages.
9. What do you understand by software development kit?
10. Differentiate SDI and MDI.

---

# UNIT 2 PROGRAMMING RESOURCE

---

## Structure

- 2.0 Introduction
- 2.1 Objectives
- 2.2 Accelerators
- 2.3 Bitmaps
- 2.4 Dialog Boxes
- 2.5 Icons
- 2.6 Menus
- 2.7 String Tables
- 2.8 Toolbars
- 2.9 Summary
- 2.10 Terminal Questions

---

## 2.0 INTRODUCTION

---

Programming resources are the important and powerful part of Windows programming; which are used to create various controls. These are associated with an important file called *rc.exe* or *resource script compiler*. The resource compiler compiles a special type of file known as a *Resource Script*. Resource scripts contain GUI data, and, when compiled, can be linked into a program. The program then can access the data contained in the resource script. Moreover, it is interesting to note that the operating system accesses a program's resource for various purposes. For instance, right-click on a program and click "Properties". If available, click on the "Version" tab which contain a number of different text strings that describe the program. These text strings are all included from a resource script.

There are several types of resources that can be stored in a program via a resource script. Resource scripts are not the only way to store this information in a program, but also they are much easier to write than hard C code or VB code to store and access them.

They are of following types:

- Drop-down Menus
- Popup Menus
- Text Strings

- Keyboard Accelerators (keypress combinations, such as [Ctrl]+[C] to copy text)
- Icons
- Bitmap Images
- Dialog Boxes
- Version Information
- Mouse Cursors

The syntax to write a resource script is similar to C code. The resource script compiler uses the standard C preprocessor and the header file `<afxres.h>` to make a resource script must also be included. Once a resource is stored in the executable file, various methods can be used to access it. It also depends on type of resource one want to access. For example, one might need to use the *LoadString* function; accordingly, the *LoadIcon* function would be needed to access an icon.

All the resources are stored using a string or number. If it is numeric, the number must be no larger than an unsigned 16-bit integer (65535, max). Resources are all called by name, that is the system is expecting a Unicode string with the name of the resource. However, if we use a numerical identifier we need to inform the system that we are doing so, so that it couldn't get confused and try to treat the integer as a string. For that the `MAKEINTRESOURCE` macro to make is passed.

---

## 2.1 OBJECTIVES

---

At the end of this unit you will come to know about:

- Accelerators
- Bitmaps
- Dialog Boxes
- Icons
- Menus
- String Tables
- Toolbars
- Versions

---

## 2.2 ACCELERATORS

---

In Windows programming, *accelerators* are the keyboard shortcuts. These are the most commonly used during the time of working in editors like MSWord, Notepad, etc. For example, Ctrl+s and Ctrl+o is used to save and open a file respectively. Most often, programs use keyboard accelerators to duplicate the action of common menu options, but they can also perform nonmenu functions.

For instance, some Windows programs have an edit menu that includes a delete or clear option; these programs conventionally assign the Del key as a keyboard accelerator for this option. The user can choose the Delete option from the menu by pressing an Alt-key combination or can use the keyboard accelerator simply by pressing the Del key. When the window procedure receives a WM\_COMMAND message, it does not have to determine whether the menu or the keyboard accelerator was used.

In the same way other shortcuts are also available in a table which is known as *accelerator table*. In Windows programming, an accelerator table allows an application to specify a list of accelerators for menu items or other commands. An accelerator takes precedence over normal processing and can be a convenient way to program some event handling. Each accelerator is associated with a control ID, which is similar to the IDs of buttons, text boxes, check boxes, and menu items. In this way, GUI objects can be created which represent the same function as an accelerator.

Since using the menus, and subsequently the mouse, is not always the good practice, it is important to provide users with the possibility to minimize usage of the mouse. Therefore, showing the accelerators in menus can be useful; it informs the user that there are shortcuts, and that using the mouse is not always mandatory. Moreover, many of the latest supercomputers are based on accelerators, including the two fastest systems according to the 11/2013 TOP500 list. Accelerators are also becoming widespread in PCs and are even starting to appear in handheld devices, which will further boost the interest in accelerator programming.

---

## 2.2.1 ACCELERATOR TABLE

---

An accelerator table can be defined in Developer Studio. For ease in loading the accelerator table in the program, give it the same text name as your program. An *accelerator table* is a data resource that maps keyboard combinations, such as Ctrl+o, to application commands. These tables provide a better solution with following characteristics:

- Requires less coding.
- Consolidates all of your shortcuts into one data file.
- Supports localization into other languages.
- Enables shortcuts and menu commands to use the same application logic.

Moreover, each shortcut in the table is defined by:

- **A numeric identifier-** It is an integer number that identifies the application command that will be invoked by the shortcut.
- **The ASCII character** or virtual-key code of the shortcut.
- **Optional modifier keys-** Alt, Shift, or, Ctrl.

The accelerator table itself has a numeric identifier, which identifies the table in the list of application resources.

---

## 2.2.2 SOME RULES ON ASSIGNING ACCELERATORS

---

In theory, you can define a keyboard accelerator for almost any virtual key or character key in combination with the Shift key, Ctrl key, or Alt key. However, you should try to achieve some consistency with other applications and avoid interfering with Windows use of the keyboard. You should avoid using Tab, Enter, Esc, and the Spacebar in keyboard accelerators because these are often used for system functions. The most common use of keyboard accelerators is for items on the program's Edit menu. The recommended keyboard accelerators for these items changed between Windows 3.0 and Windows 3.1, so it's become common to support both the old and the new accelerators, as shown in Table 2.1.

**Table 2.1** Old Accelerators and New Accelerators

Function	Old Accelerator	New Accelerator
Undo	Alt+Backspace	Ctrl+Z
Cut	Shift+Del	Ctrl+X
Copy	Ctrl+Ins	Ctrl+C
Paste	Shift+Ins	Ctrl+V
Delete or Clear	Del	Del

---

## 2.3 BITMAPS

---

A bitmap is an array of bits, where one or more bits correspond to each display pixel. In a monochrome bitmap, each pixel requires one bit. In simple case 1 bit represents white, while 0 bit represents black. Bitmaps are mostly used in logical operations rather than to create simple drawings. In a color bitmap, multiple bits correspond to each pixel to represent color. Image editor usually supports the creation of monochrome bitmaps and 16-color bitmaps.

In computer graphics, when the domain is a rectangle (indexed by two coordinates) a bitmap gives a way to store a binary image, that is, an image in which each pixel is either black or white (or any two colors). The more general term *pixmap* refers to a map of pixels, where each one may store more than two colors, thus using more than one bit per pixel. Often *bitmap* is used for this as well. In some contexts, the term *bitmap* implies one bit per pixel, while *pixmap* is used for images with multiple bits per pixel.

The great thing about MS Windows is that unlike DOS, you don't need to know anything about what video hardware you are using to display graphics. Instead, it provides an API called the Graphics Device Interface (GDI). The GDI uses a set of generic graphics objects that can be used to draw to the screen, to memory, or even to printers.

A bitmap is the representation of a picture or another type of graphics on a window. It is one of the GDI objects that can be selected into a *device context* (DC). The device contexts are structures that define a set of graphic objects and their associated attributes, and graphic modes that affect output. The table below describes the GDI objects that can be selected into a device context. A *bitmap* is a graphical object used to create, manipulate (scale, scroll, rotate, and paint), and store images as files on a disk. This overview describes the bitmap classes and bitmap operations. Fig. 2.1 illustrates the use of bitmap as a regular picture.



**Fig. 2.1 Use of bitmap as regular picture on a form**

From a developer's view, a bitmap consists of a collection of structures that specify or contain the following elements:

- A header that describes the resolution of the device on which the rectangle of pixels is created, the dimensions of the rectangle, the size of the array of bits, and so on.
- A logical palette.
- An array of bits that defines the relationship between pixels in the bitmapped image and entries in the logical palette.

A bitmap size is related to the type of image it contains. Bitmap images can be either monochrome or color. In an image, each pixel corresponds to one or more bits in a bitmap. Monochrome images have a ratio of 1 bit per pixel (bpp). Color imaging is more complex. The number of colors that can be displayed by a bitmap is equal to two raised to the number of bits per pixel. Thus, a 256-color bitmap requires 8 bpp ( $2^8 = 256$ ).

Control Panel applications are examples of applications that use bitmaps. When you select a background (or wallpaper) for your desktop, actually a bitmap is selected, which the system uses to paint the desktop background. The system creates the selected background pattern by repeatedly drawing a 32-by-32-pixel pattern on the desktop.

There are two classes of bitmaps:

- **Device-Independent Bitmaps (DIB)** - The DIB file format was designed to ensure that bitmapped graphics created using one application can be loaded and displayed in another application, retaining the same appearance as the original.
- **Device-Dependent Bitmaps (DDB)** – This is also known as GDI bitmaps, were the only bitmaps available in early versions of 16-bit Microsoft Windows (prior to version 3.0). However, as display technology improved and as the variety of display devices increased, certain inherent problems surfaced which could only be solved using DIBs. For example, there was no method of storing (or retrieving) the resolution of the display type on which a bitmap was created, so a drawing application could not quickly determine whether a bitmap was suitable for the type of video display device on which the application was running.

---

### 2.3.1 THE BITMAP CLASS

---

To support bitmaps, the GDI+ library provides the *Bitmap* class. In the .NET Framework, the bitmap is represented by a class called *Bitmap*. The *Bitmap* class is derived from the *Image* abstract class. Both classes are defined in the *System::Drawing* namespace of the *System.Drawing.dll* assembly. The *Bitmap* class is serializable.

---

### 2.3.2 THE BITMAP APPLICATIONS

---

- It can be used to show a regular picture on a form.
- It is widely used in control panel applications.
- It is used on some dialog boxes.
- A bitmap can be used as a background for a window or a web page.

#### CHECK YOUR PROGRESS

- What do you mean by accelerators?
- Write the old and new accelerators for copy and paste.
- Give three examples of *Bitmap* applications.

---

## 2.4 DIALOG BOXES

---

A dialog box is a user interface element, a type of window that is used to enable communication and interaction between a user and a software program. The dialog box appears when the program either needs to give the user information in an urgent manner that involves interruption, such as when an error occurs; or if the program requires immediate input or decision from the user such



as when the program closes and needs to know if the changes made have to be saved or not. The simplest form of dialog box is an alert, which simply displays a message and only requires an acknowledgment from the user that the message has been read as shown in Fig. 2.2.

Dialog boxes usually take onscreen priority and are displayed over the current display window. Dialog boxes also appear in response to the selection of a menu option. For example, if the user wants to change the properties of an image in image editing software, a dialog box often appears that will allow the user to select what's required to achieve the desired effect on the image. Some dialogue boxes are predefined that support common menu items such as *Open* and *Print*.

Dialogue boxes are basically used to receive inputs from the keyboard by the users which is beyond what can be managed with the menu available in the editor. A dialogue box is basically available in the form of a pop up window including different child controls. The size and placement of these controls are normally indicated in a *dialogue box template* in the program's resource script file. All Windows are responsible for the creation of these pop up dialog boxes and child controls. The code for this type of working is called *dialogue box manager*. The message processed by the dialogue box window passed to the function is called *dialogue procedure* or simply *procedure*.

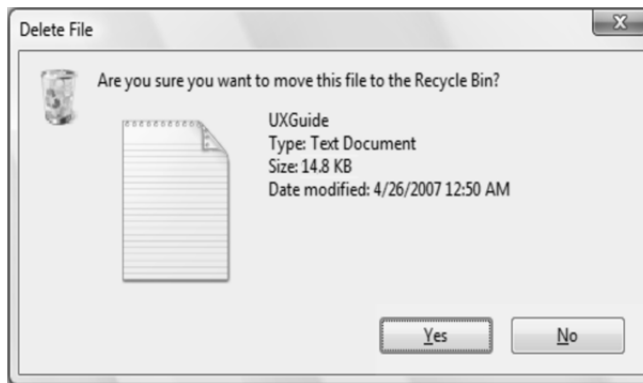
The subject of the dialogue boxes is a big one as it involves the child window controls. Adding a dialogue box to a window program is not an easy task because it involves various changes to many files like – the dialogue box template goes in the resource script file, the dialog box procedure goes in the source code file, and the identifiers used in the dialog box go in the program's header file. There are two types of dialog boxes i.e. modal and modeless.

---

### 2.4.1 MODAL DIALOG BOX

---

Modal dialog box is very common. It halts the application so that the user cannot continue until the dialog box has been closed or its purpose satisfied. This type of dialog box is used when the program needs information or is providing a warning. When the program displays this modal dialog box the user can't switch between the dialog box and another window in the program.



**Fig. 2.2 A Simple dialogue box**

---

## 2.4.2 MODELESS DIALOG BOX

---

*Modeless dialog boxes* are used when the information or action being requested by the dialog box is not essential for the proper function of the program so it can be left open while the user continues to work on the application. *Modeless dialog boxes* stay on the screen and are available for use at any time but permit other user activities. These dialog boxes are preferred when the user would find it convenient to keep the dialog box displayed for a while. For instance, word processors often use modeless dialog boxes to Find and Replace the specific text. Creating a dialog box for a program requires the following steps:

1. Use the *dialog editor* to design the dialog box and create its dialog-template resource.
2. Create a *dialog class*.
3. Connect the *dialog resource's controls to message handlers* in the dialog class.
4. Add data members associated with the dialog box's controls and to specify dialog data exchange and *dialog data validations* for the controls.

---

## 2.5 ICONS

---

An *icon* is a small graphical representation of a program or file. When you double-click an icon, the associated file or program is opened. For example, if you double-click on the My Computer icon, it would open Windows Explorer. Icons are a component of GUI operating systems including Apple macOS X and Microsoft Windows. Icons help users quickly identify the type of file represented by the icon as shown in Fig. 2.3.



**Fig. 2.3 Some icons on the desktop of computer**

The tool used to create icons, cursors, and bitmaps is called an image editor, and it is one of the most important development tools in any Windows integrated development environment. Icons and cursors are both variations of bitmaps. Basically, icons are pictorial representations of objects, these are important not only for artistic reasons as part of the visual identity of a program, but also for utilitarian reasons as shorthand for conveying meaning that users

perceive almost instantaneously. Windows introduces a new style of iconography that brings a higher level of details and sophistication for users.

Moreover, in computing, an icon is a pictogram or ideogram displayed on a computer screen in order to help the user navigate a computer system. The icon itself is a quickly comprehensible symbol of a software tool, function, or a data file, accessible on the system and is more like a traffic sign than a detailed illustration of the actual entity it represents. It can serve as an electronic *hyperlink* or *file shortcut* to access the program or data. The user can activate an icon using a mouse, pointer, finger, or recently voice commands. Their placement on the screen, also in relation to other icons, may provide further information to the user about their usage. In activating an icon, the user can move directly into and out of the identified function without knowing anything further about the location or requirements of the file or code.

There are numerous software programs available that enable users to customize, create, or modify their own icons on their computers. A suggestion of some of the free programs is given below:

- [Free icon editor](#)
- [aaICO](#)
- [LiquidIcon](#)
- [IcoFx](#)

---

## **2.5.1 TYPES OF ICONS**

### **2.5.1.1 STANDARDIZED ELECTRICAL DEVICE SYMBOLS**

---

A series of recurring computer icons are taken from the broader field of standardized symbols used across a wide range of electrical equipment. For examples the power symbol and the USB icon, which are found on a wide variety of electronic devices. The standardization of electronic icons is an important safety-feature on all types of electronics, enabling a user to more easily navigate an unfamiliar system. As a subset of electronic devices, computer systems, and mobile devices use many of the same icons; they are incorporated into the design of both the computer hardware and on the software. On the hardware, these icons identify the functionality of specific buttons and plugs. In the software, they provide a link into the customizable settings.

System warning icons also belong to the broader area of ISO standard warning signs. These warning icons, first designed to regulate automobile traffic in the early 1900s, have become standardized and widely understood by users without necessity of further verbal explanations. In designing software operating systems, different companies have incorporated and defined these standard symbols as part of their GUI. For example, the Microsoft MSDN defines the standard icon use of error, warning, information and question mark icons as part of their software development guidelines.

Different organizations are actively involved in standardizing these icons, as well as providing guidelines for their creation and use. The International Electrotechnical Commission (IEC) has defined "Graphical symbols for use on

equipment," published as IEC 417, a document which displays IEC standardized icons. Another organization invested in the promotion of effective icon usage is the Information and Communications Technologies (ICT), which has published guidelines for the creation and use of icons. Many of these icons are available on the Internet, either to purchase or as freeware to incorporate into new software.

---

### **2.5.1.2 METAPHORICAL ICONS**

---

An icon is a Signifier pointing to a Signified. Easily comprehensible icons will make use of familiar visual metaphors directly connected to the Signified: actions the icon initiate or the content that would be revealed. Metaphors, Metonymy and Synecdoche are used to encode the meaning in an icon system. The Signified can have multiple natures: virtual objects such as Files and Applications, actions within a system or an application (e.g. snap a picture, delete, rewind, connect/disconnect etc...), action in the physical world (e.g. print, eject dvd, change volume or brightness etc...) as well as physical objects (e.g. monitor, compact disk, mouse, printer etc...).

---

### **2.5.1.3 THE DESKTOP METAPHOR**

---

A subgroup of the more visually rich icons is based on objects lifted from a 1970 physical office space and desktop environment. It includes the basic icons used for a file, file folder, trash can, inbox, together with the spatial real estate of the screen i.e. the electronic desktop. This model originally enabled users, familiar with common office practices and functions, to intuitively navigate the computer desktop and system. The icons stand for objects or functions accessible on the system and enable the user to do tasks common to an office space. These desktop computer icons developed over several decades; data files in the 1950s, the hierarchical storage system (i.e. the file folder and filing cabinet) in the 1960s, and finally the desktop metaphor itself in the 1970s.

Dr. David Canfield Smith associated the term "icon" with computing in his landmark 1975 PhD thesis *"Pygmalion: A Creative Programming Environment."* In his work, Dr. Smith envisioned a scenario in which "visual entities", called icons, could execute lines of programming code, and save the operation for later re-execution. Dr. Smith later served as one of the principal designers of the Xerox Star, which became the first commercially available personal computing system based on the desktop metaphor when it was released in 1981.

This model of the desktop metaphor has been adopted by most personal computing systems in the last decades of the 20th century; it remains popular as a "simple intuitive navigation by single user on single system." It is only at the beginning of the 21st century that personal computing is evolving a new metaphor based on Internet connectivity and teams of users, cloud computing. In this new model, data and tools are no longer stored on the single system, instead they are stored someplace else, "in the cloud". The cloud metaphor is replacing the desktop model; it remains to be seen how many of the common desktop icons (file, file folder, trash can, inbox, filing cabinet) find a place in this new metaphor.

---

### 2.5.1.4 BRAND ICONS FOR COMMERCIAL SOFTWARE

---

A further type of computer icon is more related to the brand identity of the software programs available on the computer system. These brand icons are bundled with their product and installed on a system with the software. They function in the same way as the hyperlink icons described previously, representing functionality accessible on the system and providing links to either a software program or data file. Over and beyond this, they act as a company identifier and advertiser for the software or company. Because these company and program logos represent the company and product itself, much attention is given to their design, done frequently by commercial artists. To regulate the use of these brand icons, they are trademark registered and are considered part of the company intellectual property.

---

### 2.5.1.5 OVERLAY ICONS

---

In some GUI systems (e.g. Windows), on an icon which represents an object (e.g. a file) a certain additional subsystem can add a smaller secondary icon, layer over the primary icon and usually positioned in one of its corners, to indicate the status of the object which is represented with the primary icon. For instance, the subsystem for locking files can add a "padlock" overlay icon on an icon which represents a file in order to indicate that the file is locked.

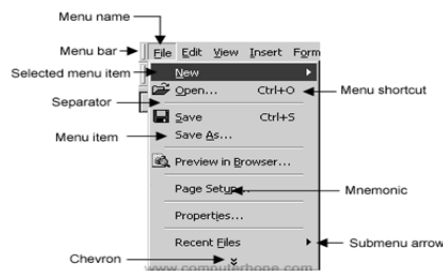
---

## 2.6 MENUS

---

A menu is a list of available options. A menu tells a hungry person what a restaurant can serve for him/her. A window's menu bar is displayed below the title bar. This menu bar is called *main menu* which is also called *popup menu* or a *submenu*. An item in a popup menu can display another menu and so on. Sometimes options/items in a menu displays dialogue boxes instead of further menus.

Alternatively referred to as the *file menu*, a *menu* is a list of commands or choices offered to the user through the menu bar. Menus are commonly used in GUI operating systems and allow a user to access various options the software program is capable of performing. File menus are commonly accessed using the computer mouse. However, it may also sometimes be accessed using shortcuts or the keyboard. Fig. 2.4 is a visual illustration of what a menu may look like in a GUI environment.



**Fig. 2.4 Visual Example of Menu**  
(Source: [www.computerhope.com](http://www.computerhope.com))

For a window program a menu tells the user that what type of operations an application can perform. A menu is basically very important part of any user interface that a window can offer. Adding a menu is easy way in Windows programming. To create a menu, simply design a structure in the resource script and assign a unique ID to each menu item. And the name of the menu can be defined in the window class structure. When the user chooses a menu item, window sends a WM\_COMMAND message containing that ID for the program.

Menu items in the top level menu or in popup menus can be *enabled*, *disabled*, or *grayed*. The words active and inactive are also used for enabled and disabled respectively. Enabled and disabled items look same but have a slightly different gray shade. The functions of all these three are different. Window sends WM\_COMMAND message to program only for enabled messages.

---

## 2.6.1 MOST COMMONLY USED MENUS

### 2.6.1.1 PROPERTIES MENU

---

Properties menu is the important feature of Windows programming and normally a user is familiar with it. It is helpful to navigate on other options. It is available by clicking the right click on any icon and displays several properties of the particular file or directory. In Microsoft Windows, a properties page is a panel of information in the file properties dialog (accessed from the File menu, context menu, or by typing alt + Enter, or alt + double-clicking). It can be a built-in feature of Windows Explorer (for example, the file sharing page), or created by a shell extension (for example MP3ext or WinRAR).

---

### 2.6.1.2 SYSTEM MENU

---

The system menu (also called the window menu or control menu) is a popup menu in Microsoft Windows, accessible by left-clicking on the upper-left icon of most windows, or by pressing the Alt and Space keys. This menu provides the user ability to perform some common tasks on the window, some in unusual ways. For example, normally a user would move a window by dragging the title bar of the window - but with the option in the system menu the user gets a different cursor and procedure to move the window with.

Some applications customize the system menu, typically through the GetSystemMenu WinAPI function. The Win32 console, used e.g. by the Command Prompt (cmd.exe), is an example of this and it offers the user an ability to change its preferences through its system menu (other applications typically offer the user to change their preferences through the normal menu below their window's title bar).

---

## 2.6.2 MENU STRUCTURE

---

When menu is created or changed in a program, it is useful to think about the top-level menu and each popup menu must be kept as a separate menu. The top-level menu has a menu handle, each popup menu within a top-level menu has its own menu handle. Each item in the menu is defined by three different features.

The first feature is what appears in the menu which is either a text string or a bitmap. The second feature is either an ID number that windows send to the program in a WM\_COMMAND message or a popup menu that windows displays when the user selects that menu item. And the third one describes the attribute of the menu item, including whether the item is disabled, grayed, or checked.

### CHECK YOUR PROGRESS

- What do you mean by modal dialog box?
- Define modeless dialog box.
- Discuss about the menu structure.

---

## 2.7 STRING TABLES

---

String tables are programming resources to hold text strings. They can be added or edited by resource editor present in Visual Studio. String tables are basically a list of strings which is compiled and gets added in executable binary. These strings are in UNICODE format and thus can support all international languages like Hindi, English, Chinese, Arabic, Hebrew, Japanese, Spanish, French, etc. *LoadString API* call is an important way which can be used to obtain string from string table. LoadString basically loads a string resource from the executable file associated with a specified module, copies the string into a buffer, and appends a terminating null character.

A resource script can have many string tables, although this is unnecessary: the tables aren't differentiated (i.e. they get merged), and each string object, in any table, must have a unique identifier. Strings in a string table also may not use names, but instead must use numeric identifiers. After all, it doesn't make any sense to have to address a string with a string. String tables are used to store string variables that are referenced by workflows for runtime replacement of values, depending on the environment in which the workflow is executing. When a workflow executes and references one of these placeholders, the workflow checks the string table to find the value for the placeholder and replaces the "placeholder" with the string value for that string table entry. The character string resources are defined in resource script using keyword STRINGTABLE:

```
STRINGTABLE
{
    id1, "character string 1"
    id2, "character string 2"
    [other string definitions]
}
```

**Note:** - The resource script can contain only one String table. Each string can be only one line long with a maximum of 255 characters.

---

## 2.7.1 STRING TABLE EDIT FIELDS

---

The right-hand side of the String Table Editor window displays a series of fields for editing the selected string. The first field, the ID field, is where you can modify the string ID name, which is the name associated with each string ID. This name will be included in your string file as an enumerated list, and you will use this name in your application software when you want to refer to a particular string. You can edit this name simply by typing on the keyboard. You can select the font to use while working in the string table using the drop-down list box labeled Font. The selected font is displayed in the grid in the lower-right portion of the screen. This grid allows you to select characters while editing the current string. The second field on the right side of the String Table Editor window is the string literal edit window. This field displays the string literal value using any of the fonts which are part of your project.

There are three methods for editing strings displayed in the string literal edit window. The *First* one, if the current language alphabet is supported by your keyboard, you can simply type the string value. In *Second* method, you can simply click on characters displayed in the font viewer window, as you click on the characters, they are inserted into the current string at the current insertion point. Finally, you can type the *JIS (Japanese Industrial Standard)* or *Unicode* encoding value in hexadecimal for the character you wish to insert. For people who know the encodings for common characters, this is faster than finding the characters in the font display window. As you edit the selected string, the width of the string (in pixels) is displayed in the Width field. The Notes button brings up a small note editor window. Notes are useful for including additional information about each string, usually for the benefit of translators who will translate your English or reference language strings into strings for the other languages.

The reference language is also important in the event that a translation is not required or available for certain strings in your application. Empty (NULL) strings in any other language are replaced with the corresponding (untranslated) string of the reference language.

---

## 2.7.2 MERGING STRING TABLES

---

The Merge button on the String Table Editor window invokes a series of dialogs that walk through the merge process. In order to understand the reason for the merge operation, we need to examine the life-cycle of a typical multi-language project development as given below:

1. The system developers define the initial string table. The total number of languages and the language names are defined using the Configure | Languages dialog.
2. The String ID names and the Reference Language (English) strings are initialized for all strings in the application using the String Table Editor.
3. The Window Builder Project file, along with the Window Builder executable program, are distributed to translators who will each fill in



one column of the string table. These translators may reside at the same location, but could also be located all around the globe.

4. The translators return Window Builder project files to you, and the returned project files each have one or more additional columns of the string table filled in with translated strings.

The problem should now be obvious: The Merge operation will merge strings for selected languages from a second project file into the current project file. The process is actually very simple as you are guided step-by-step through the merge process. When Window Builder performs the merge, it looks for matching string ID names in the secondary project. For each matching string ID name, if the selected language in the secondary project has a non-NULL string value, that string value is copied into the current project for that specific string ID and language.

---

## 2.8 TOOLBARS

---

Toolbars serve as an always-available, easy-to-use interface for performing common functions. A *toolbar* is a set of icons or buttons that are part of a software program's interface or an open window. When it is part of a program's interface, the toolbar typically sits directly under the menu bar. For example, Adobe Photoshop includes a toolbar that allows you to adjust settings for each selected tool. If the paintbrush is selected, the toolbar will provide options to change the brush size, opacity, and flow. MS Word has a toolbar with icons that allow you to open, save, and print documents, as well as change the font, text size, and style of the text. Like many programs, the Word toolbar can be customized by adding or deleting options. It can even be moved to different parts of the screen.

The toolbar can also reside within an open window. For example, web browsers, such as Internet Explorer, include a toolbar in each open window. These toolbars have items such as Back and Forward buttons, a Home button, and an address field. Some browsers allow to customize the items in toolbar by right-clicking within the toolbar and choosing "Customize..." or selecting "Customize Toolbar" from the browser preferences. Open windows on the desktop may have toolbars as well. For example, in Mac OS X, each window has Back and Forward buttons, View Options, a Get Info button, and a New Folder button. You can customize the Mac OS X window toolbars as well.

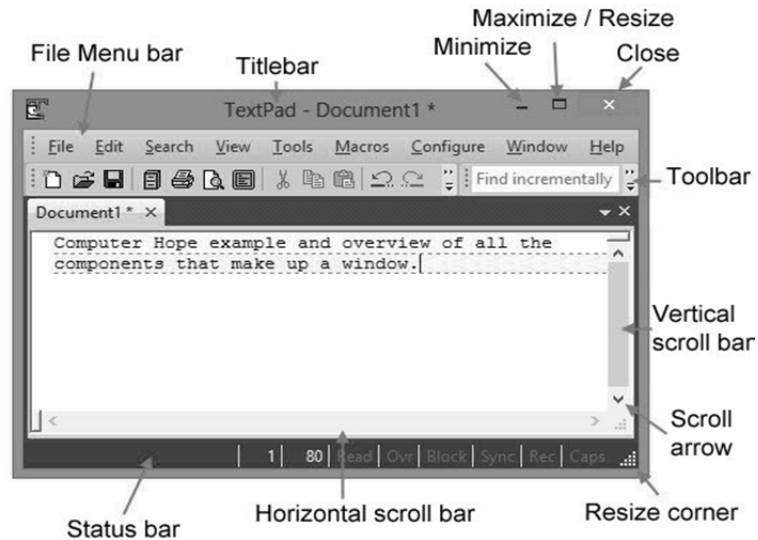
Sometimes referred to as a bar or *standard toolbar*, the toolbar is a row of boxes, often at the top of an application window that control various functions of the software. The boxes often contain images that correspond with the function they control, as demonstrated in the image below. A toolbar often provides quick access to functions that are commonly performed in the [program](#). For example, a [formatting toolbar](#) in a [MS Excel](#) gives you access to things like making text **bold** or changing its alignment, along with other common [buttons](#). In an [Internet browser](#), toolbars add functionality that may not come pre-installed. For example, with the Google toolbar, you can get access to exclusive Google features.

**Note:** - If you are missing a window toolbar, press the *Alt* key on the keyboard as

some programs hide the toolbar until Alt is pressed.

There are several common Computer Software Toolbars and other Bars listed as follows.

- **App Bar** - Windows 8 bar.
- **Bar Chart** - A chart consisting of horizontal or vertical bars.



**Fig. 2.5 Example of Microsoft window toolbar, menu bar, title bar**

- **Barcode** - A series of lines that identifies an address, product, or other information.
- **Browser Toolbar** - Any toolbar in an Internet browser.
- **Bookmarks Bar** - Bar showing frequently visited bookmarks or favourites.
- **Commands Bar** - A bar that shows available commands in a program.
- **Formatting Toolbar** - Toolbar that shows text formatting options.
- **Formula Bar** - Bar in a spreadsheet program that allows you to edit a formula.
- **Menu Bar** - A bar at the top of the screen that gives access to all of the menus.
- **Navigation Bar** - Gives access to all navigation features in a browser.
- **Places Bar** - A pane that shows common places to access files.
- **Progress Bar** - An indicator that shows how long until something is completed.
- **Scroll Bar** - A bar on the bottom or side of the window to scroll through a page.

- **Split Bar** - A bar that divides the window into multiple sections.
- **Status Bar** - One of the few bars at the bottom of the window that shows the status.
- **Title Bar** - A bar at the very top of a window that describes the program or window.

Apart from the software toolbars, there are some computer Hardware Bars too, for example:

- **Port Bar** - A device that allows your laptop to connect to other devices.
- **Spacebar** - A key on a keyboard that creates a space.

### CHECK YOUR PROGRESS

- What do you understand by string table?
- Write any three examples of toolbars.

---

## 2.9 SUMMARY

---

In Windows programming, *accelerators* are the keyboard shortcuts. These are commonly used during the time of working in most of the editors like MSWord, and Notepad, etc. For example, Ctrl + s and Ctrl + o, are used to save and open a file respectively. Most often, programs use keyboard accelerators to duplicate the action of common menu options, but they can also perform non-menu functions.

A *bitmap* is an array of bits where one or more bits corresponds to each display pixel. In a monochrome bitmap, each pixel requires one bit. In simple case, 1 represents white while 0 represents black. Bitmaps are mostly used in logical operations rather than to create simple drawings. In a color bitmap, multiple bits are used for each pixel to represent colors. Image editor usually supports the creation of monochrome bitmaps and 16-color bitmaps.

A *dialog box* is a user interface element, a type of window that is used to enable communication and interaction between a user and a software program. The dialog box appears when the program either needs to give the user information in an urgent manner that involves interruption, such as when an error occurs, or if the program requires immediate input or decision from the user, such as when the program closes and needs to know if the changes made have to be saved or not.

An *icon* is a small graphical representation of a program or file. When you double-click an icon, the associated file or program is opened. For example, if you double-click on the My Computer icon, it would open Windows Explorer. Icons are a component of GUI operating systems, including Apple macOS and Microsoft Windows. Icons help users quickly identify the type of file represented by the icon.

A *menu* is a list of available options. A window's menu bar is displayed below the title bar. This menu bar is called main menu which is also called *popup menu* or a *submenu*. An item in a popup menu can display another menu and so on. Some popular menus are- file menu, edit menu, and font menu.

*String tables* are programming resources to hold text strings. They can be added or edited by resource editor present in Visual Studio. String tables are basically a list of strings which are compiled and get added in executable binary. These strings are in UNICODE format and thus can support all international languages like Hindi, English, Chinese, Arabic, and Hebrew, etc.

A *toolbar* is a set of icons or buttons that are part of a software program's interface or an open window. When it is the part of a program's interface, the toolbar typically sits directly under the menu bar. For example, Adobe Photoshop includes a toolbar that allows you to adjust settings for each selected tool. If the paintbrush is selected, the toolbar will provide options to change the brush size, opacity, and flow.

---

## 2.10 TERMINAL QUESTIONS

---

1. What do you understand by accelerators? Explain its usefulness.
2. Draw the table for all available old and new accelerators.
3. Explain bitmap class and its types.
4. Compare modal and modeless dialog boxes.
5. Write a short note on types of icons.
6. Discuss most commonly used menus.
7. What do you understand by string table? Explain merging of string tables.
8. Write a short note on common computer software toolbars and other bars.

---

# UNIT-3 VISUAL C++ PROGRAMMING

---

## Structure

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Basic Concepts of VC++
- 3.3 Object Oriented Programming
- 3.4 Object and Classes
- 3.5 VC++ Components
- 3.6 Resources
- 3.7 Event Handling
- 3.8 Menus
- 3.9 Dialog Boxes
- 3.10 MFC File Handling
- 3.11 MFC and VC++
- 3.12 Summary
- 3.13 Terminal Questions

---

## 3.0 INTRODUCTION

---

Microsoft Visual C++ (often abbreviated to MSVC) is an integrated development environment (IDE) product from Microsoft for the C, C++, and C++/CLI programming languages. MSVC is a proprietary software. It was originally a standalone product but later became a part of Visual Studio and made available in both trialware and freeware forms. It features tools for developing and debugging C++ code, especially code written for the Windows API, DirectX and .NET.

Many applications require redistributable Visual C++ runtime library packages to function correctly. These packages are often installed independently of applications, allowing multiple applications to make use of the package while only having to install it once. The Visual C++ (VC++) is redistributable and runtime packages are mostly installed for standard libraries that many applications use. It has tools for coding and debugging visual codes. The main features of VC++ are:

- Smart pointers
- New containers

- Expression parsing
- Polymorphism
- Exception handling
- Garbage collection

In this unit, the programming using VC++ is discussed with basic as well as some advanced features.

---

### 3.1 OBJECTIVES

---

At the end of this unit you are supposed to acquire knowledge about the following:

- Object Oriented Programming
- Objects
- Classes
- VC++ Components
- Resources
- Event Handling
- Menus
- Dialog Boxes
- MFC file handling
- MFC and VC++

---

### 3.2 BASIC CONCEPTS OF VC++

---

VC++ in its version 1.5 was released in December 1993. It included MFC 2.5, and added OLE 2.0 and ODBC support to MFC. It was the first version of VC++ that came only on CD-ROM. MSVC is an IDE product from Microsoft for the C, C++, and C++/CLI programming languages. It includes tools for developing and debugging C++ code, especially code written for the Windows API, DirectX and .NET. Many applications require redistributable VC++ runtime library packages to function correctly. These packages are often installed independently of applications, allowing multiple applications to make use of the package while only having to install it once. These VC++ redistributable and runtime packages are mostly installed for standard libraries that many applications use. You can use it in an integrated development system or as individual tools. VC++ consists of the following components:

- **VC++ Compiler Tools-** The compiler supports traditional native code developers and also the developers who target virtual machine platforms such as the common language runtime (CLR). The compiler evolved over the years through different versions such as 2008, 2010, and 2012, etc.

VC++ 2008 includes compilers to target x64 and Itanium. The compiler continues to support targeting x86 computers directly, and optimizes performance for both platforms.

- **VC++ Libraries-** These include the industry-standard Active Template Library (ATL), the Microsoft Foundation Class (MFC) libraries, standard libraries such as the Standard C++ Library consisting of the iostreams library, Standard Template Library (STL), and the C Runtime Library (CRT). The CRT includes security-enhanced alternatives to functions that are known to pose security issues. The STL/CLR library brings STL to managed developers. The C++ Support Library is included with new features for data marshaling to simplify programs that target the CLR.
- **VC++ Development Environment-** It provides powerful support for project management and configuration (including better support for large projects), source code editing, source code browsing, and debugging tools. This environment also supports IntelliSense, which makes informed, context-sensitive suggestions as code is being authored.

VC++ comes within Microsoft Visual Studio. And usually, Visual Studio contains Visual Basic, Visual C#, and Visual J#. Using Visual Studio, you can mix and match languages within one "solution". Visual Studio is a programming environment that contains all the libraries, examples, and documentation needed to create applications for Windows. Instead of talking about programs, in Visual Basic we talk about projects and solutions. Solutions can contain several projects and projects typically contain multiple items or files.

---

## 3.2.1 CREATING FIRST VC++ PROGRAM

---

For the first VC++ program, a console mode application is made that displays a greeting message. A console mode application is a kind of VC++ program that is built for all exercises/assignments. Console mode programs are often simpler to build than Windows applications. The following example will take you through the steps of creating, building and executing a program in VC++. We first assume that you use the built-in code editor in Visual Studio to edit your code; then we will show you how to build and run your C++ programs that you have created with any external editors.

---

### 3.2.1.1 HOW TO START

---

You can open the Visual Studio by following the steps given below.

*Press Start | All Programs | Microsoft Visual Studio 2013 | Visual Studio 2013*

You may also find a shortcut to Visual Studio 2013 on your desktop.

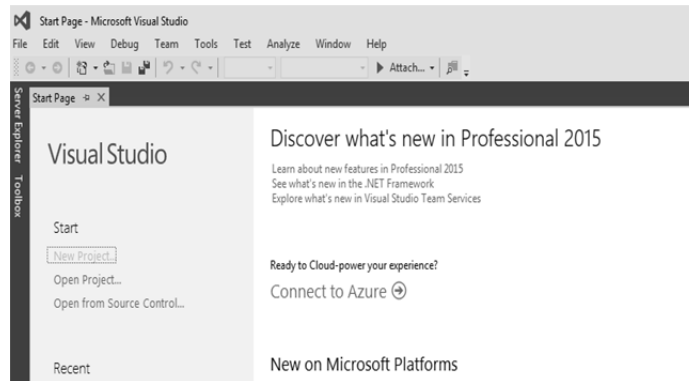
---

### 3.2.1.2 STARTING YOUR FIRST PROGRAM

---

If Visual Studio is being used first time, you have to select VC++ Development Settings, then click on Start Visual Studio. If you want to choose another environment later, to the menu and click *Tools\Import and Export\Reset*

All Settings\Next, then you choose to save current settings or overwrite current settings. After that you will see is the Start Page as shown in Fig. 3.1.

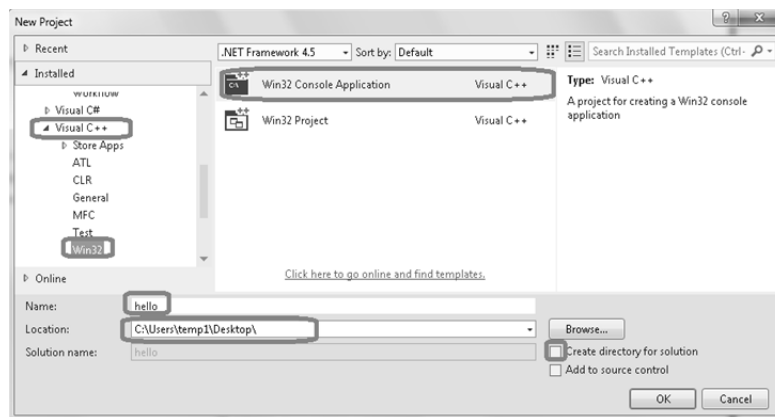


**Fig. 3.1 Start Page**

To get started on your first program, you must create a "project" that will keep track of all the parts of your program, including the C++ source code, and the header files, etc. Therefore, click the "Create Project" link. A "New Project" dialog box similar to the Fig. 3.2 would appear. Now follow the steps given as follows:

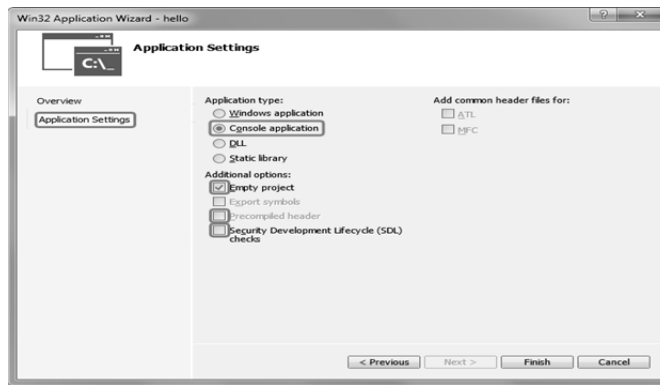
- For a *Name*, type a project name (e.g. "hello")
- *Location*, set it to your Desktop, or to your *Documents\Visual Studio 2013\Projects*
- Click on *OK*

The "Win32 Application Wizard" would appear as shown in Fig. 3.3, click on *Application Settings*, uncheck both *Precompiled Header* and *Security Development Lifecycle (SDL)* then select *Empty Project*. After this, click on *Finish*. You will notice that it doesn't appear like anything has changed. However, if you look at the *Solution Explorer* on the right-hand side you will see "Solution 'hello' (1 project)".



**Fig. 3.2 New Project Dialog Box**





**Fig. 3.3 Win32 Application Wizard**

To add C++ source code to this project, use following steps:

- Click on your project in the *Solution Explorer*
- Select *Project | Add New Item...* from the main menu
- Select *C++ File (.cpp)*
- Type in the file name: "hello.cpp" in the *Name:* box.
- Click on *Add*.

This file will be added to the *hello* workspace that we have just created, and a blank document will be opened for editing.

**Exercise:** *Type the following program in the source code editor and Save hello.cpp after you have finished editing it.*

```
// FILE: hello.cpp
```

```
// PURPOSE: An example of a simple I/O stream
```

```
#include <iostream>
using namespace std;

int main()
{
    char name[50];
    cout << "Please enter your name:" << endl;

    cin >> name;
    cout << "Hello, " << name << endl;
    return 0;
}
```

---

### 3.2.1.3 BUILDING THE HELLO PROJECT

---

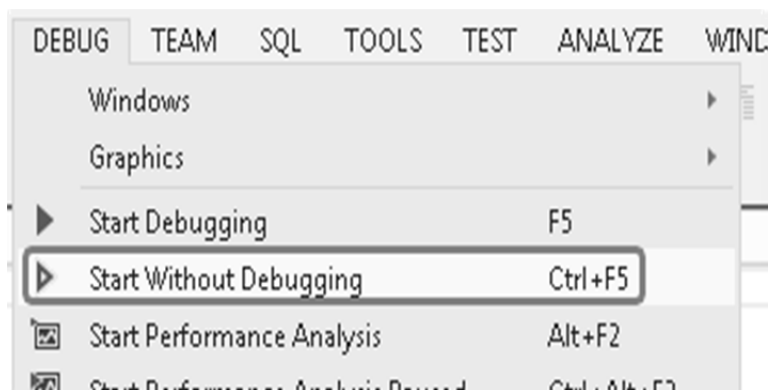
In order to compile any code in VC++, you have to create a project first. A project holds three major types of information:

- It remembers all of the source code files that combine together to create one executable. In this simple example, the file `hello.cpp` will be the only source file, but in larger applications you often break the code up into several different files to make it easier to understand (and also to make it possible for several people to work on it simultaneously). The project maintains a list of the different source files and compiles all of them as necessary each time you want to create a new executable.
- It remembers compiler and linker options particular to this specific application. For example, it remembers which libraries to link into the executable, whether or not you want to use pre-compiled headers, and so on.
- It remembers what type of project you wish to build: a console application, a windows application, etc.

If you are familiar with makefiles, then it is easy to think of a project as a machine-generated makefile that has a very easy-to-understand user interface to manipulate it.

#### Exercise:

- Compile the hello project by selecting *BUILD | Compile* from the main menu.  
It simply compiles the source file and forms the object file (`hello.obj`) for it. It does not perform a link, so it is useful only for quickly compiling a file to check for errors.
- Select *BUILD | Build hello* from the menu bar to link the program. It compiles all of the source files in the project that have been modified since the last build, and then links them to create an executable.
- Choose *DEBUG | Start Without Debugging* to run the program.



**Fig. 3.4 Debugging**

If errors or warnings are displayed in the Build status window, there is probably an error in the source file. Check your source file again for missing semicolons, quotes, or braces. Now, we will continue using VC++ in another project, so select *File | Close Solution*. This will return you to the Start Page.

---

### 3.2.2 ADVANTAGE OF OOP OVER PROCEDURE-ORIENTED PROGRAMMING

---

1. OOPs makes development and maintenance easier whereas in Procedure-Oriented programming it is not easy to manage if code grows as project size grows.
2. OOPs provides data hiding whereas in Procedure-Oriented programming a global data can be accessed from anywhere.
3. OOPs provide ability to simulate real-world event much more effectively. We can provide the solution to real word problem if we are using the Object-Oriented Programming.

---

## 3.3 OBJECT ORIENTED PROGRAMMING

---

Object Oriented programming is a programming style that is associated with the concept of class, objects and various other concepts revolving around these two like inheritance, polymorphism, abstraction, encapsulation, and data binding, exception handling etc. The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.

Object Oriented Programming is also called a paradigm that provides the concepts such as encapsulation, inheritance, abstraction etc. The programming paradigm where everything is represented as an object is known as truly or pure object-oriented programming language like Java; on the other hand, the language that does not represent everything as an object is not pure object oriented language like C++. In other words, the program in C++ can be created with or without a class. *Smalltalk* is considered as the first truly object-oriented programming language.

Now, let us try to understand a little about all these, through a simple real life example. Human beings are living forms, broadly categorized into two types, Male and Female. It is true that every human being has two legs, two hands, two eyes, one nose, one heart etc. There are body parts that are common for male and female, but some specific body parts, present in a male are not present in a female; and some body parts present in female but not in males. All human beings can walk, eat, see, talk, hear etc. Now again, both male and female, performs some common functions, but there are some specifics to both, which is not valid for the other. For example: A female can give birth, while a male cannot, so this is only for the female. Human anatomy is an interesting area. But let us see how all this is related to C++ and OOPS. Here we will try to explain all the OOPS concepts through this example and later we will have the technical definitions for all this.

---

### 3.3.1 OBJECT ORIENTED PROGRAMMING CONCEPTS

---

**Class** - Here we can consider *Human Being* as a class. A class is a blueprint for any functional entity which defines its properties and its functions. Like Human Being, having body parts, and performing various actions. Moreover, it is similar to structure and union in C language. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declares & defines what data variables the object will have and what operations can be performed on the class's object.

**Object** - My name is Krishan, and I am an instance/object of class male. When we say, Human Being, male or female, we just mean a kind, you, your friend, me, we are the forms of these classes. We have a physical existence while a class is just a logical definition. We are the objects. Object is everything for an object oriented programming. Every concept(s) is/are associated with the object. The objects are basic unit of object oriented programming. They are instances of a class, which have data members and uses various member functions to perform various tasks. These objects are modeled after the real life things like animals, flowers, birds, students etc.

**Encapsulation** - This concept is a little tricky to explain with our example. Our legs are binded to help us walk. Our hands, help us hold things. This binding of the properties to functions is called Encapsulation. From programming point of view, "*Wrapping of the data and functions into a single unit is called encapsulation.*" This is to avoid the access of private data members from outside the class. To achieve encapsulation, we make all data members of class private and create public functions, using them we can get the values from these data members or set the value to these data members. It is implemented using the class or in simple words we can say that class is the real implementation of encapsulation. It can also be said as data binding. Encapsulation is all about binding the data variables and functions together in class.

**Abstraction** - Abstraction means, showcasing only the required things to the outside world while hiding the details. Continuing our real life example of human being; human beings can talk, walk, hear, eat, but the details about how it happens; are hidden from the outside world. We can take our skin as the abstraction factor in our case, hiding the inside mechanism of sensing so many things so fast with appropriate classifications of different materials. Moreover, abstraction refers to showing only the essential features of the application and hiding the details. Abstraction is a process of hiding irrelevant details from user. For example, when a person sends an sms he/she just type the message, select the contact and click send, the phone shows that the message has been sent; but what actually happens in background when you click send, is hidden from the sender as it is considered as irrelevant.

In C++, classes can provide methods to the outside world to access & use the data variables, keeping the variables hidden from direct access, or classes can even declare everything accessible to everyone, or maybe just to the classes inheriting it. This can be done using access specifiers. Steering of the car, electrical switch are some other examples of real life abstraction. In C++ it can be

implemented using the pure virtual functions.

**Inheritance** - Considering HumanBeing a class, which has properties like hands, legs, eyes etc, and functions like walk, talk, eat, see etc. Male and female are also classes, but most of the properties and functions are included in Human Being, hence they can inherit everything from class HumanBeing using the concept of Inheritance like in their children. More conceptually, Inheritance is a way to reuse the code written once. The class which is inherited is called the base class & the class which inherits is called the derived class. They are also called parent and child class. So when, a derived class inherits a base class, the derived class can use all the functions which are defined in the base class, hence making the code reusable. It gives the benefit of reusability. There are five types of inheritance in object oriented languages viz single, multiple, hierarchical, multilevel, and hybrid.

**Polymorphism** – Polymorphisms means one thing and many uses; it allows us to redefine the way something works, by either changing how it is done or by changing the parts using which it is done. Both the ways have different terms for them. If we walk using our hands, and not legs, here we will change the parts used to perform something. Hence this is called *overloading*. And if there is a defined way of walking, but I wish to walk differently, but using my legs, like everyone else. Then I can walk like I want, this will be called as *overriding*. It lets us create functions with same name but different/same arguments as per the need, which will perform different actions. That means, functions with same name, but functioning in different ways. Or, it also allows us to redefine a function to provide it with a completely new definition. You will learn how to do this in details soon in coming lessons. It is of two types i.e. runtime and compile time polymorphism. It can be used to increase the alternatives.

**Exception Handling** - Exception handling is used to handle unresolved exceptions or errors produced at runtime. Throw and catch are two popular keywords used in exception handling. It is a good way to avoid errors in software development and hence avoiding risks of software fail. Normally try is the keyword used in many languages for throwing error in the form of object and catch to handle that object in efficient and effective manner without causing problems at runtime.

---

## 3.4 OBJECTS AND CLASSES

---

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities. A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

---

### 3.4.1 OBJECTS

---

*Object* means a real world entity such as pen, chair, table etc. as shown in Fig. 3.5. **Collection of objects** is called a class basically which is a logical entity. VC++ is a multi-paradigm programming language. Meaning, it supports different programming styles. One of the popular ways to solve a programming problem is by creating objects, known as object-oriented style of programming. VC++

supports object-oriented style of programming which allows one to divide complex problems into smaller sets by creating objects. Object is simply a collection of data and functions that act on those data.



**Fig. 3.5 Objects**

---

### **3.4.2 CLASSES**

---

The classes are the most important feature of C++ that leads to object oriented programming. Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance of that class. The variables inside class definition are called as *data members* and the functions are called *member functions*. For example- class of birds; all birds can fly and they all have wings and beaks. So, here flying is a behavior and wings and beaks are part of their characteristics. And there are many different birds in this class with different names but they all possess this behavior and characteristics. Similarly, class is just a blue print, which declares and defines characteristics and behavior, namely data members and member functions respectively. And all objects of this class will share these characteristics and behavior. Some more information on classes is given as follows.

1. Class name should start with an uppercase letter (Although this is not mandatory). If class name is made of more than one word, then first letter of each word must be in uppercase. For Example:  
class Student, class StudyTonight, class Birds etc.
2. Classes contain data members and member functions, and the access of these data members and variables depend on the access specifiers e.g. private, public, protected (which is discussed in details later.)
3. Class member functions can be defined inside the class definition or outside the class definition.
4. Classes in C++ are similar to structures in C, the only difference being, class defaults to private access control, where as structure defaults to public.
5. All the features of OOPs in VC++ e.g. Inheritance, Encapsulation, Abstraction etc. revolve around the objects and classes.
6. Objects of a class holds separate copies of data members. We can create as many objects of a class as we need.
7. Classes do possess more characteristics such as abstract classes, immutable classes etc.

## CHECK YOUR PROGRESS

- Give any two advantages of OOPs over procedure oriented programming.
- What do you understand by encapsulation?
- What do you understand by inheritance?

## 3.5 VC++ COMPONENTS

VC++ consists of two complete Windows application development systems in one product. If you choose, you can develop C-language Windows programs using only the Win32 API. You can use many VC++ tools, including the resource editors, to make low-level Win32 programming easier. VC++ also includes the ActiveX Template Library (ATL), which can be used to develop ActiveX controls for the Internet. ATL programming is neither Win32 C language programming nor MFC programming. It is about C++ programming within the MFC library application framework that is part of VC++. You'll be using the C++ classes documented in the VC++ *MFC Library Reference* and you'll also be using application framework-specific VC++ tools such as AppWizard and ClassWizard.

**Note:** - Use of the MFC library programming interface doesn't cut you off from the Win32 functions. In fact, you'll almost always need some direct Win32 calls in your MFC library programs.

A quick run-through of the VC++ components will help you to get your bearings in the application framework. Fig. 3.6 shows an overview of the VC++ application build process.

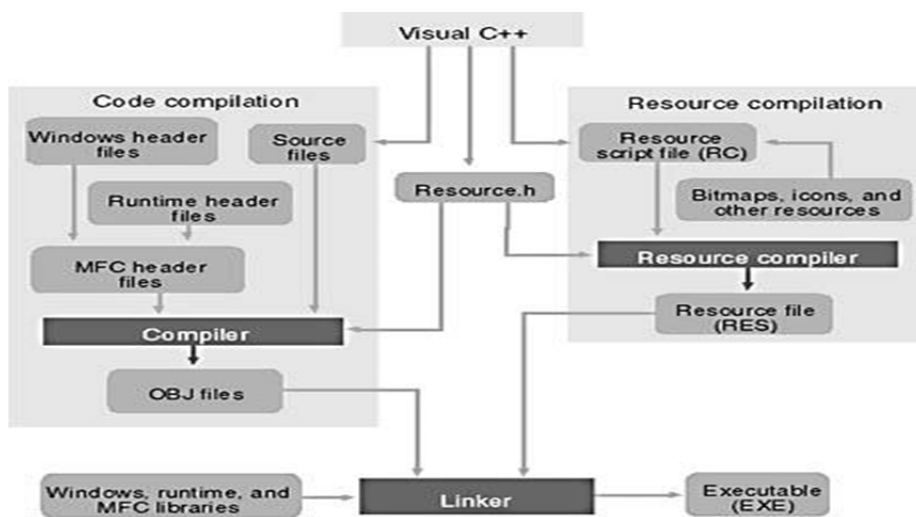


Fig. 3.6 Application Building Process in VC++

(source: <https://www.tenouk.com/visualcplum/mfc/visualcplum/mfc1.html>)

If you have used earlier versions of VC++ or another vendor's IDE, you already understand how VC++ operates. But if you are new to IDEs, you will need to know what a project is. Basically, a project is a collection of interrelated source files that are compiled and linked to make up an executable Windows-based program or a DLL. Source files for each project are generally stored in a separate subdirectory. A project depends on many files outside the project subdirectory too, such as include files and library files.

Experienced programmers are familiar with *makefiles*. A makefile stores compiler and linker options and expresses all the interrelationships among source files. A source code file needs specific *include* files, certain object modules, and libraries, and so forth. A make program reads the makefile and then invokes the compiler, assembler, resource compiler, and linker to produce the final output, which is generally an executable file. The make program uses built-in inference rules that tell it to invoke the compiler to generate an OBJ file from a specified CPP file.

In a VC++ 6.0 (or VC6 in short) project, there is no makefile (with an MAK extension) unless you tell the system to export one. A text-format project file (with a DSP extension) serves the same purpose. A separate text-format workspace file (with a DSW extension) has an entry for each project in the workspace. It is possible to have multiple projects in a workspace. To work on an existing project, you need to tell VC++ to open the DSW file and then you can edit and build the project. VC++ creates some intermediate files too. Table 3.1 lists the files that VC++ generates in the workspace.

**Table 3.1** Files in the Workspace

<b>File Extension</b>	<b>Description</b>
APS	Supports ResourceView
BSC	Browser information file
CLW	Supports ClassWizard
DEP	Dependency file
DSP	Project file*
DSW	Workspace file*
MAK	External makefile
NCB	Supports ClassView
OPT	Holds workspace configuration
PLG	Builds log file



*Note: DSP and DSW are two important file extensions used for project and workspace respectively.*

---

### **3.5.1 THE RESOURCE EDITORS—WORKSPACE RESOURCEVIEW**

---

When you click on the ResourceView tab in VC++ workspace window, you can select a resource for editing. The main window hosts a resource editor appropriate for the resource type. The window can also host a *wysiwyg editor* for menus and a powerful graphical editor for dialog boxes, and it includes tools for editing icons, bitmaps, and strings. The dialog editor allows you to insert ActiveX controls in addition to standard Windows controls and the new Windows common controls.

Each project usually has one text-format resource script (RC) file that describes the project's menu, dialog, string, and accelerator resources. The RC file also has *#include* statements to bring in resources from other subdirectories. These resources include project-specific items, such as BMP and icon (ICO) files, and resources common to all VC++ programs, such as error message strings. Editing the RC file outside the resource editors is not recommended. The resource editors can also process EXE and DLL files, so you can use the clipboard to "steal" resources, such as bitmaps and icons, from other Windows applications.

---

### **3.5.2 THE C/C++ COMPILER**

---

The VC++ compiler can process both C and C++ source code. It determines the language by looking at the source code's filename extension. A C extension indicates C source code, and CPP or CXX indicates C++ source code. The compiler is compliant with all ANSI standards, including the latest recommendations of a working group on C++ libraries, and has additional Microsoft extensions. Templates, exceptions, and runtime type identification (RTTI) are fully supported in VC++ version 6.0. The C++ Standard Template Library (STL) is also included, although it is not integrated into the MFC library.

---

### **3.5.3 THE SOURCE CODE EDITOR**

---

VC++ 6.0 includes a sophisticated source code editor that supports many features such as dynamic syntax coloring, auto-tabbing, keyboard bindings for a variety of popular editors (such as VI and EMACS), and pretty printing. An exciting new feature named AutoComplete has been added. If you have used any of the Microsoft Office products or Microsoft Visual Basic, you might already be familiar with this technology. Using the VC++ 6.0 AutoComplete feature, all you have to do is type the beginning of a programming statement and the editor would provide you with a list of possible completions to choose from. This feature is extremely handy when you are working with C++ objects and have forgotten an exact member function or data member name—they are all there in the list for you. You no longer have to memorize thousands of Win32 APIs or rely heavily on the online help system.

---

### 3.5.4 THE RESOURCE COMPILER

---

Resource Compiler (**rc**) - a tool that compiles resources such as icons, cursors, menus, and dialog boxes, that your application uses. You add the resulting binary resource file to the application's binary file to produce an executable Windows 3.x application. Moreover, it is a tool used in building Windows-based applications. This overview describes how to create a resource-definition (script) file, how to compile your application's resources, and how to add compiled resources to your application. This tool is available in Visual Studio and the Microsoft Windows Software Development Kit (SDK). It Resource compiler reads an ASCII RC file from the resource editors and writes a binary RES file for the linker.

---

### 3.5.5 THE LINKER

---

The linker reads the OBJ file produced by the C/C++ compiler, RES file generated by the resource compiler, LIB files for MFC code, runtime library code, and Windows code. It then writes the project's EXE file. An incremental link option minimizes the execution time when only minor changes have been made to the source files. The MFC header files contain *#pragma* statements (special compiler directives) that specify the required library files, so you don't have to tell the linker explicitly which libraries to read.

---

### 3.5.6 THE DEBUGGER

---

If your program executes first time, you don't need the debugger. The rest of us might need one from time to time. The VC++ debugger has been steadily improving, but it doesn't actually fix the bugs yet. The debugger works closely with VC++ to ensure that breakpoints are saved on disk. Toolbar buttons insert and remove breakpoints and control single-step execution. Note that the Variables and Watch windows can expand an object pointer to show all data members of the derived class and base classes. If you position the cursor on a simple variable, the debugger shows you its value in a small window. To debug a program, you must build the program with the compiler and linker options set to generate debugging information.

VC++ 6.0 adds a new twist to debugging with the *Edit And Continue* feature. *Edit And Continue* lets you debug an application, change the application, and then continue debugging with the new code. This feature dramatically reduces the amount of time you spend debugging because you no longer have to manually leave the debugger, recompile, and then debug again. To use this feature, simply edit any code while in the debugger and then hit the continue button. VC++ 6.0 automatically compiles the changes and restarts the debugger for you.

---

### 3.5.7 APPWIZARD

---

*AppWizard* is a code generator that creates a working skeleton of a Windows application with features, class names, and source code filenames that you specify through dialog boxes. Don't confuse AppWizard with older code

generators that generate all the code for an application. The functionality is inside the application framework base classes. AppWizard gets you started quickly with a new application. Advanced developers can build custom AppWizards. Microsoft Corporation has exposed its macro-based system for generating projects. If you discover that your team needs to develop multiple projects with a telecommunications interface, you can build a special wizard that automates the process.

---

### 3.5.8 CLASSWIZARD

---

*ClassWizard* is a program (implemented as a DLL) that is accessible from VC++ View menu. ClassWizard takes the hard work out of maintaining VC++ class code. ClassWizard writes the prototypes, the function bodies, and the code to link the Windows message to the function (if necessary). ClassWizard can update class code that you write, so you avoid the maintenance problems common to ordinary code generators.

---

### 3.5.9 THE SOURCE BROWSER

---

If you write an application from scratch, you probably have a good mental picture of your source code files, classes, and member functions. If you take over someone else's application, you'll need some assistance. The VC++ Source Browser (the browser, for short) lets you examine (and edit) an application from the class or function viewpoint instead of from the file viewpoint. It is a little like the "inspector" tools available with object-oriented libraries such as Smalltalk. The browser has the following viewing modes:

1. *Definitions and References*: You select any function, variable, type, macro, or class and then see where it's defined and used in your project.
2. *Call Graph/Callers Graph*: For a selected function, you see a graphical representation of the functions it calls or the functions that call it.
3. *Derived Classes and Members/Base Classes and Members*: These are graphical class hierarchy diagrams. For a selected class, you see the derived classes or the base classes plus members. You can control the hierarchy expansion with the mouse.
4. *File Outline*: For a selected file, the classes, functions, and data members appear together with the places in which they're defined and used in your project.

---

### 3.5.10 ONLINE HELP

---

In VC++ 6.0, the help system has been moved to a separate application named the MSDN Library Viewer. This help system is based on HTML. Each topic is covered in an individual HTML document, then all topics are combined into indexed files. The MSDN Library Viewer uses code from Microsoft Internet Explorer 4.0, so it works like the Web browser you already know. MSDN Library can access the help files from the VC++ CD-ROM (the default installation option)

or from your hard disk, and it can access HTML files on the Internet. VC++ 6.0 allows to access help in four ways:

1. *By book:* When you choose Contents from Help menu, the MSDN Library application switches to a contents view. Here Visual Studio, VC++, Win32 SDK documentation, and more is organized hierarchically by books and chapters.
2. *By topic:* When you choose Search from VC++ Help menu, it automatically opens the MSDN Library Viewer. You can then select the Index tab, type a keyword, and see the topics and articles included for that keyword.
3. *By word:* When you choose Search from VC++ Help menu, it invokes the MSDN Library with the Search tab active. With this tab active, you can type a combination of words to view articles that contain those words.
4. *F1 help:* This is the programmer's best friend. Just move the cursor inside a function, macro, or class name, and then press the F1 key and the help system goes to work. If the name is found in several places—in the MFC and Win32 help files, for example—you choose the help topic you want from a list window.

---

### 3.5.11 WINDOWS DIAGNOSTIC TOOLS

---

VC++ 6.0 contains a number of useful diagnostic tools. The SPY++ tool gives you a tree view of your system's processes, threads, and windows. It also lets you view messages and examine the windows of running applications. You find PVIEW (PVIEW95 for Windows 95) tool useful for killing errant processes that aren't visible from the Windows 95 task list. The Windows NT Task Manager, which you can run by right-clicking the toolbar, is an alternative to PVIEW. VC++ also includes a whole suite of ActiveX utilities, an ActiveX control test program (now with full source code in VC++ 6.0), the help workshop (with compiler), a library manager, binary file viewers and editors, a source code profiler, and other utilities.

---

### 3.5.12 SOURCE CODE CONTROL

---

During development of VC++ 5.0, Microsoft bought the rights to an established source code control product named *SourceSafe*. This product has since been included in the Enterprise Edition of VC++ and Visual Studio Enterprise, and it is integrated into VC++ so that one can coordinate large software projects. The master copy of the project's source code is stored in a central place on the network, and programmers can check out modules for updates. These checked-out modules are usually stored on the programmer's local hard disk. After a programmer checks in modified files, other team members can synchronize their local hard disk copies to the master copy. Other source code control systems can also be integrated into VC++.

---

### 3.5.13 THE GALLERY

---

The VC++ Components and Controls Gallery lets you share software components among different projects. The Gallery manages three types of modules:

- *ActiveX Controls*: When you install an ActiveX control (OCX—formerly OLE control), an entry is made in the Windows Registry. All registered ActiveX controls appear in the Gallery's window, so you can select them in any project.
- *Source Modules*: When you write a new class, you can add the code to the Gallery. The code can then be selected and copied into other projects. You can also add resources to the Gallery.
- *Components*: The Gallery can contain tools that let you add features to your project. Such a tool could insert new classes, functions, data members, and resources into an existing project. Some component modules are supplied by Microsoft (Idle time processing, Palette support, and Splash screen, for example) as part of VC++. Others will be supplied by third-party software firms.

---

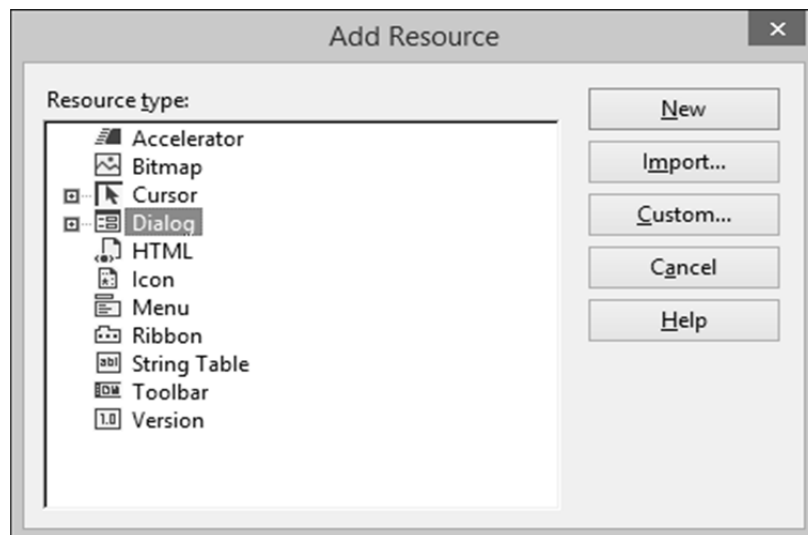
## 3.6 RESOURCES

---

Apart from framework all other objects in a windows are treated as resources. A separate .rc file holds the description of the resource. Resource compiler which is required to built into VC++ IDE and called automatically when a project has resources included. A *resource* is a text file that allows the compiler to manage objects such as pictures, sounds, mouse cursors, dialog boxes, etc. Microsoft Visual Studio makes creating a resource file particularly easy by providing the necessary tools in the same environment used to program. This means, you usually do not have to use an external application to create or configure a resource file.

Following are some important features related to resources:

- Resources are interface elements that provide information to the user.
- Bitmaps, icons, toolbars, and cursors are all resources.
- Some resources can be manipulated to perform an action such as selecting from a menu or entering data in dialog box.
- An application can use various resources that behave independently of each other, these resources are grouped into a text file that has the \*.rc extension.
- Most resources are created by selecting the desired one from the Add Resource dialog box.
- The Add Resource dialog box (Fig. 3.7) provides an extensive list of resources which can be used as per requirements, but if you need something which is not available then you can add it manually to the \*.rc file before executing the program.



**Fig. 3.7 Adding Resource**

---

### 3.6.1 IDENTIFIERS

---

An *identifier* is a symbol which is a constant integer whose name usually starts with ID. It consists of two parts – a text string (symbol name) mapped to an integer value (symbol value).

- Symbols provide a descriptive way of referring to resources and user-interface objects, both in your source code and while you're working with them in the resource editors.
- When you create a new resource or resource object, the *resource editors* provide a default name for the resource, for example, IDC\_DIALOG1, and assign a value to it.
- The name-plus-value definition is stored in the Resource.h file.
- 3.6.2 Icons
- An *icon* is a small picture used on a window which represents an application. It is used in two main scenarios.
- On a window's frame, it is displayed on the left side of the window name on the title bar.
- In Windows Explorer, on the desktop, in My Computer, or in the Control Panel window.

---

### 3.6.3 TOOLBARS

---

A *toolbar* is a Windows control that allows the user to perform some actions on a form by clicking a button instead of using a menu. It has the following features:

- A toolbar provides a convenient group of buttons that simplifies the user's job by bringing the most accessible actions as buttons.

- A toolbar can bring such common actions closer to the user.
- Toolbars usually display under the main menu.
- They can be equipped with buttons but sometimes their buttons or some of their buttons have a caption.
- Toolbars can also be equipped with other types of controls.

---

## 3.7 EVENT HANDLING

---

An *event* is a message sent by an object within a program to the main program loop, informing it that something has happened. An application is made of various objects. Most of the time, more than one application run on the computer and the operating system is constantly asked to perform some assignments. Because there can be so many requests presented unpredictably, the operating system leaves it up to the objects to specify what they want, when they want it, and what behavior or result they expect. Event handling is necessary because of the following:

- The Microsoft Windows operating system cannot predict what kinds of requests one object would need to take care of and what type of assignment another object would need.
- To manage all these assignments and requests, the objects send messages.
- Each object has the responsibility to decided what message to send and when.
- In order to send a message, a control must create an event.
- To make a distinction between the two, a message name usually starts with WM\_ which stands for Window Message.
- The name of an event usually starts with symbols that indicate an action.
- The event is the action of sending the message.
- Since Windows is a message-oriented operating system, a large portion of programming for the Windows environment involves message handling. Each time an event such as a keystroke or mouse click occurs, a message is sent to the application, which must then handle the event.
- For the compiler to manage messages, they should be included in the class definition.
- The *DECLARE\_MESSAGE\_MAP* macro should be provided at the end of the class definition as shown in the following code.

```
class CMainFrame : public CFrameWnd {
public:
    CMainFrame();
protected:
    DECLARE_MESSAGE_MAP()
};
```

- The actual messages should be listed just above the `DECLARE_MESSAGE_MAP` line.
- To implement the messages, you need to create a table of messages that your program is using.
- This table uses two delimiting macros; Its starts with a `BEGIN_MESSAGE_MAP` and ends with an `END_MESSAGE_MAP` macros.

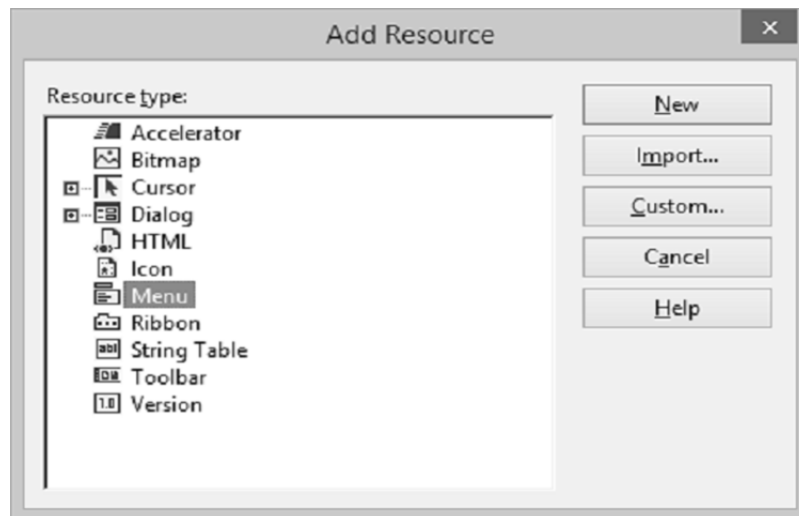
### CHECK YOUR PROGRESS

- Define class wizard.
- What do you understand by a resource?
- Discuss the role of events in Windows programming.

## 3.8 MENUS

*Menus* allow you to arrange commands in a logical and easy-to-find fashion. With the Menu editor, you can create and edit menus by working directly with a menu bar that closely resembles the one in your finished application. To create a menu, follow the steps given below

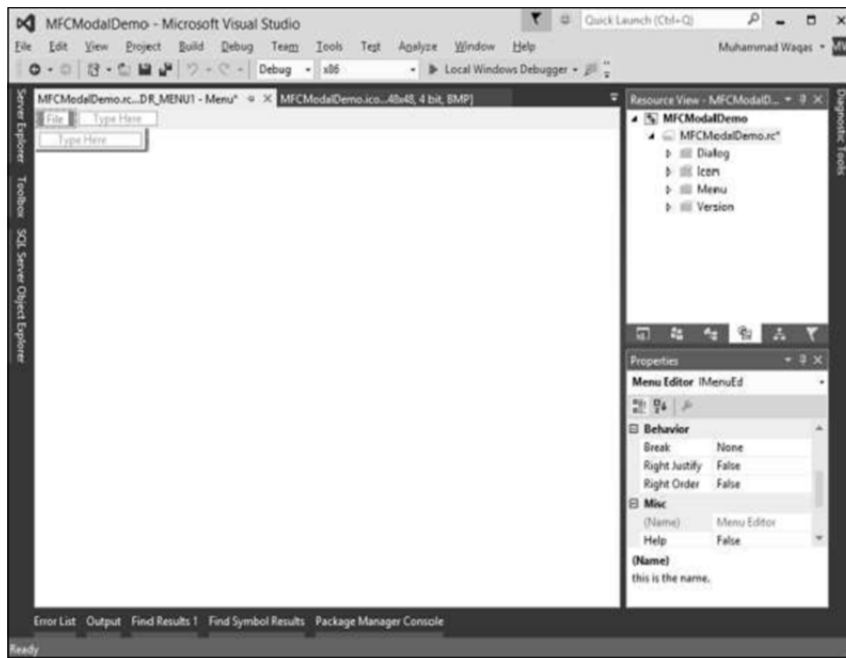
**Step 1**– Right-click on your project and select Add → Resources. You will see the Add Resources dialog box (Fig. 3.8).



**Fig. 3.8 Adding Resource**

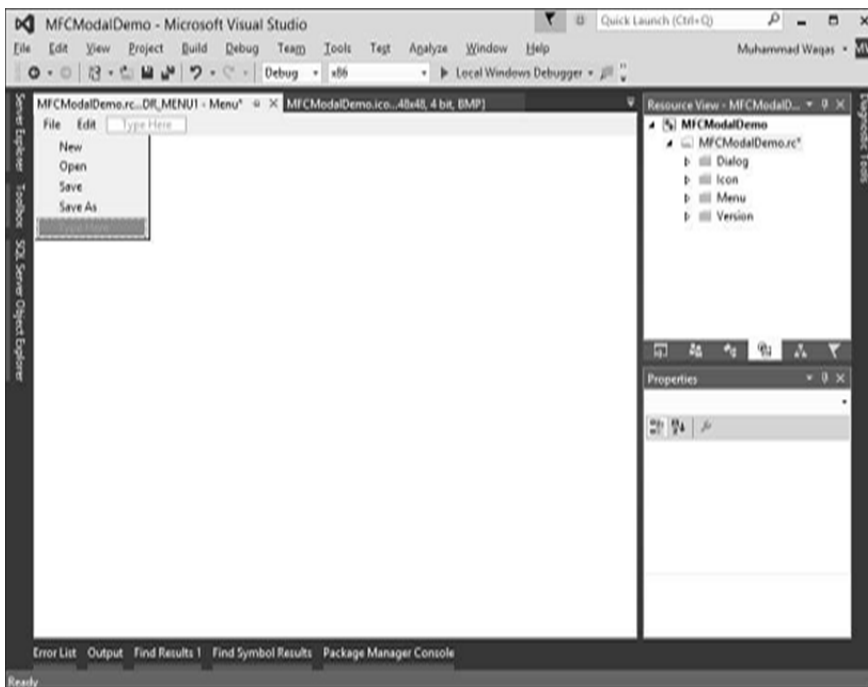
**Step 2**– Select Menu and click New. You will see the rectangle that contains "Type Here" on the menu bar (Fig 3.9).





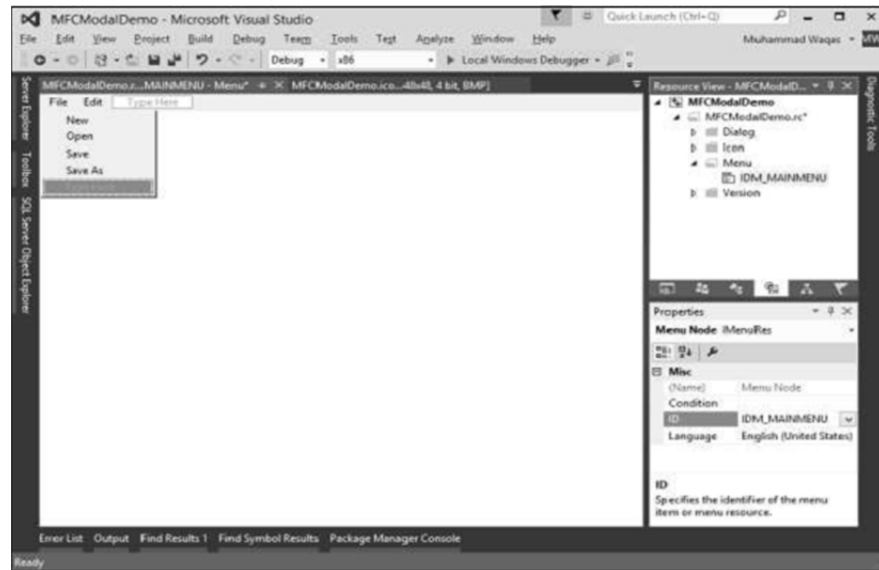
**Fig. 3.9 Type Here on the Menu Bar**

**Step 3**– Write some menu options like File, Edit, etc. as shown in the following snapshot. (Fig. 3.10)



**Fig. 3.10 Menu Options**

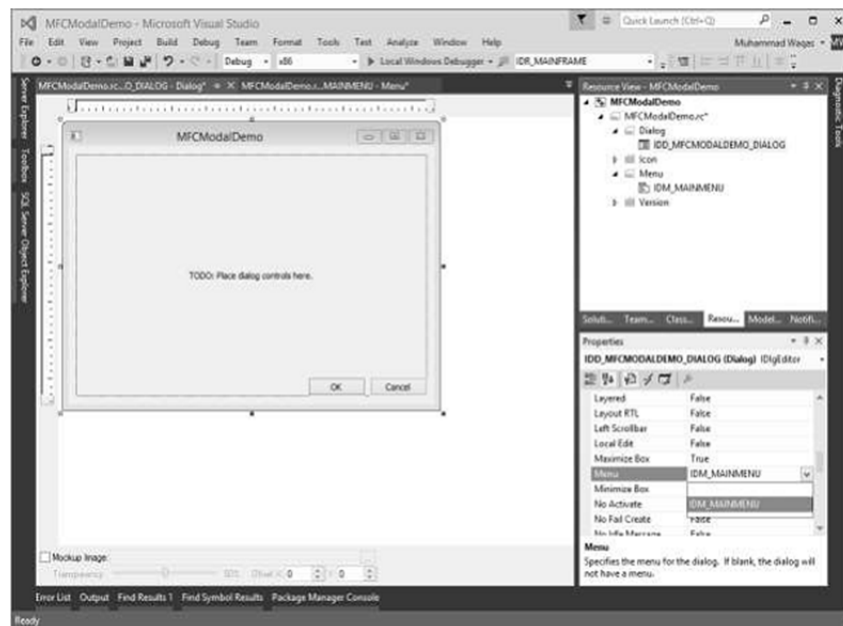
**Step 4**– If you expand the Menu folder in Resource View, you will see the Menu identifier IDR\_MENU1. Right-click on this identifier and change it to IDM\_MAINMENU. (Fig. 3.11)



**Fig. 3.11 Menu Identifier**

**Step 5**– Save all the changes.

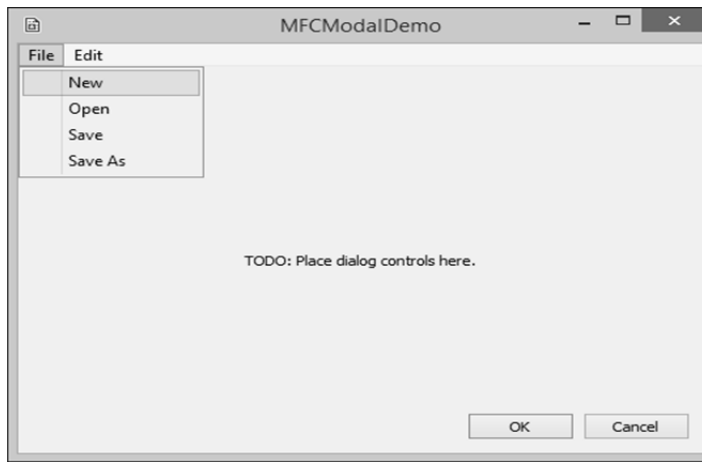
**Step 6**– We need to attach this menu to our dialog box. Expand your Dialog folder in Solution Explorer and double click on the dialog box identifier.



**Fig. 3.12 MFCModalDemo**

**Step 7**– You will see the menu field in the Properties. Select the Menu identifier from the dropdown as shown above (Fig. 3.12).

**Step 8**– Run this application and you will see the following dialog box (Fig. 3.13) which also contains menu options.



**Fig. 3.13 Dialog Box Containing Menu Options**

### 3.9 DIALOG BOXES

Applications for Windows frequently communicate with the user through dialog boxes. *CDialog* class provides an interface for managing dialog boxes. The VC++ dialog editor makes it easy to design dialog boxes and create their dialog-template resources.

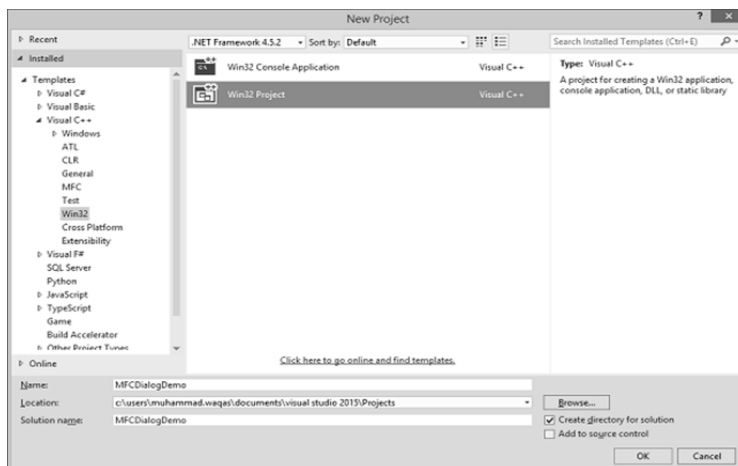
Creating a dialog object is a two-phase operation –

- Construct the dialog object.
- Create the dialog window.

Let us look into a simple example by creating a new Win32 project.

**Step 1**– Open the Visual studio and click on the File → New → Project menu option.

**Step 2**– You can now see the New Project dialog box.



**Fig. 3.14 New Project**

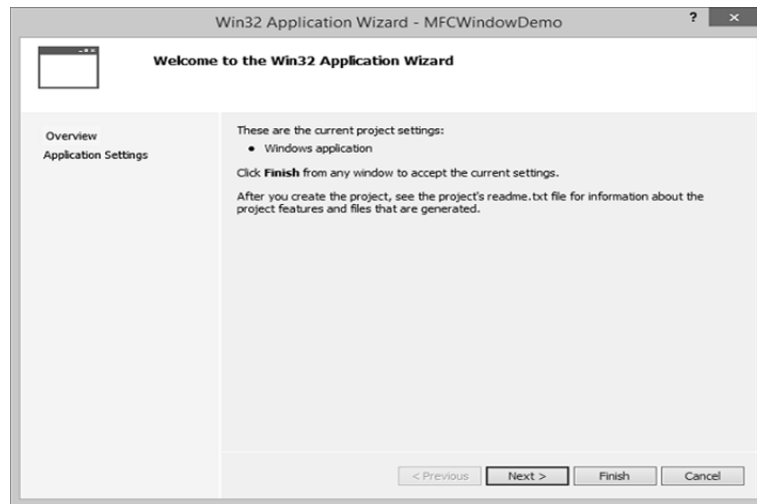
**Step 3**– From the left pane, select Templates → VC++ → Win32.

**Step 4**– In the middle pane, select Win32 Project.

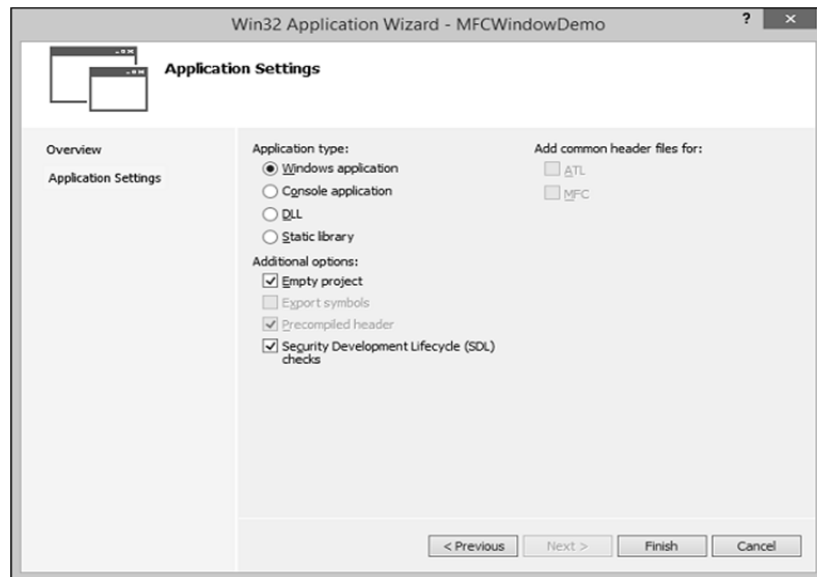
**Step 5**– Enter project name ‘MFCDialogDemo’ in the Name field and click OK to continue.

You will see the following dialog.

**Step 6**– Now Click Next.



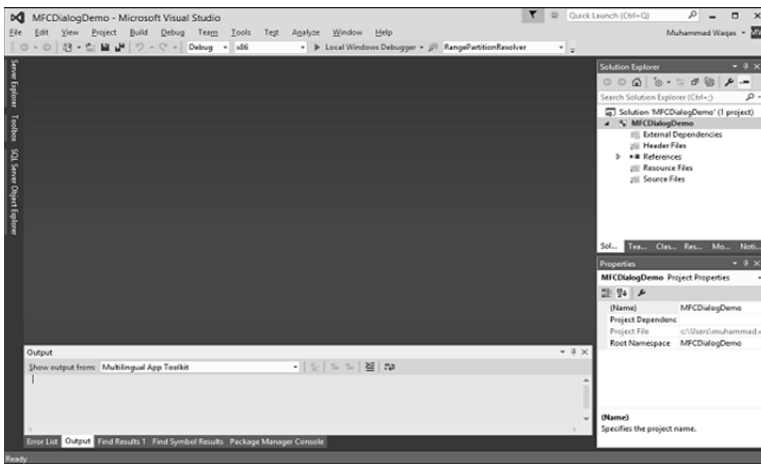
**Fig. 3.15 (a) Application Wizard**



**Fig. 3.15 (b) Application Wizard**

**Step 7**– Select the options shown in the dialog box given above and click Finish.

**Step 8**– An empty project is created (Fig. 3.16).

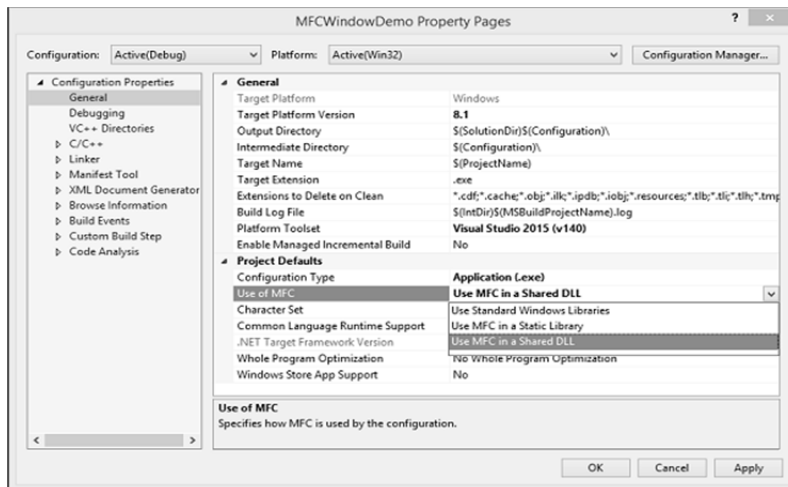


**Fig. 3.16 Empty Project**

**Step 9**– To make it a MFC project, right-click on the project and select Properties.

**Step 10**– In the left section, click Configuration Properties → General.

**Step 11**– Select the Use MFC in Shared DLL option in Project Defaults section and click OK.



**Fig. 3.17 Property Pages**

**Step 12**– Add a new source file.

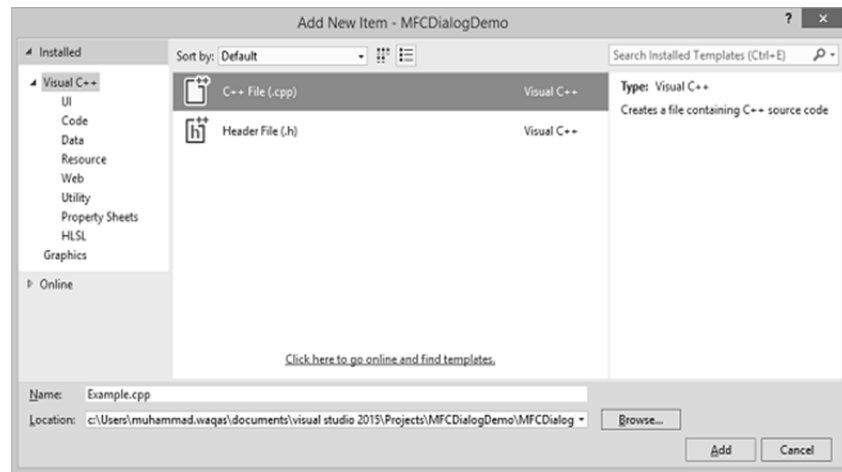
*(For Step 13-15, See Fig 3.18)*

**Step 13**– Right-click on your Project and select Add → New Item.

**Step 14**– In the Templates section, click C++ File (.cpp)

**Step 15**– Set the Name as Example and click Add.

**Step 16**– To create an application, we need to add a class and derive it from the MFC's CwinApp.



**Fig. 3.18 Add New Item**

## 3.10 MFC FILE HANDLING

The Microsoft Foundation Class (MFC) Library is a collection of classes (generalized definitions used in object-oriented programming) that can be used in building application programs. The classes in the MFC Library are written in the C++ programming language. The MFC Library saves a programmer time by providing code that has already been written. It also provides an overall framework for developing the application program. There are MFC Library classes for all GUI elements (windows, frames, menus, tool bars, status bars, and so forth), for building interfaces to databases, for handling events such as messages from other applications, for handling keyboard and mouse input, and for creating ActiveX controls.

### 3.10.1 DRIVES

A *drive* is a physical device attached to a computer so it can store information. A logical disk, logical volume or virtual disk (vdisk for short) is a virtual device that provides an area of usable storage capacity on one or more physical disk drive(s) in a computer system. A drive can be a hard disk, a CD ROM, a DVD ROM, a flash (USB) drive, a memory card etc. One of the primary operations you might want to perform is to get a list of drives on the computer.

Let us look into a simple example by creating a new MFC dialog based application.

**Step 1**– Drag one button from the toolbox, change its Caption to Get Drives Info.

**Step 2**– Remove the Caption of Static control (TODO line) and change its ID to IDC\_STATIC\_TEXT.

**Step 3** – Right-click on the button and select Add Event Handler.

**Step 4**– Select the BN\_CLICKED message type and click the Add and Edit button.

**Step 5**– Add the value variable m\_strDrives for Static Text control.

To support drives on a computer, the Win32 library provides the *GetLogicalDrives()* function of Microsoft Windows, which retrieves a list of all drives on the current computer.

**Step 6**– When the above code is compiled and executed, you see the following output.

**Step 7** – When you click the button, you can see all the drives on your computer.

---

### 3.10.2 DIRECTORIES

---

In computing, a *directory* is a file system cataloging structure which contains references to other computer files, and possibly other directories. Directory is a physical location. It can handle operations not available on a drive. Let us look into a simple example by creating a new MFC dialog based application

**Step 1**– Drag three buttons from the toolbox. Change their Captions to Create Directory, Delete Directory, and Move Directory.

**Step 2**– Change the IDs of these buttons to IDC\_BUTTON\_CREATE, IDC\_BUTTON\_DELETE, and IDC\_BUTTON\_MOVE.

**Step 3**– Remove the TODO line.

**Step 4**– Add event handler for each button.

**Step 5**– To create a directory, you can call the *CreateDirectory()* method of the Win32 library.

**Step 6**– There is a Create button event handler implementation in which we will create one directory and then two more sub directories.

**Step 7**– To get rid of a directory, you can call the *RemoveDirectory()* function of the Win32 library. Here is the implementation of delete button event handler.

**Step 8**– If you want to move a directory, you can also call the same *MoveFile()* function. Here is the implementation of move button event handler in which we will create first new directory and then move the Dir2 to that directory.

**Step 9**– When the above code is compiled and executed, output will be displayed.

**Step 10**– When you click the Create Directory button, it will create these directories.

**Step 11**– When you click on Delete Directory button, it will delete the Dir1.

---

### 3.10.3 FILE PROCESSING

---

The most of the *file processing* in an MFC application is performed in conjunction with a class named *CArchive*. The *CArchive* class serves as a relay between the application and the medium used to either store data or make it available. It allows you to save a complex network of objects in a permanent binary form (usually disk storage) that persists after those objects are deleted.

---

## 3.11 MFC AND VC++

---

MFC is a large and extensive C++ class hierarchy that makes Windows application development significantly easier. MFC is compatible across the entire Windows family. As each new version of Windows comes out, MFC gets modified so that old code compiles and works under the new system. MFC also gets extended, adding new capabilities to the hierarchy and making it easier to create complete applications.

The advantage of MFC and C++ as opposed to directly accessing the Windows API from a C program is that MFC already contains and encapsulates all the normal "boilerplate" code that all Windows programs written in C must contain. Programs written in MFC are therefore much smaller than equivalent C programs. On the other hand, MFC is a fairly thin covering over the C functions, so there is little or no performance penalty imposed by its use. It is also easy to customize things using the standard C calls when necessary since MFC does not modify or hide the basic structure of a Windows program. The best part about using MFC is that it does all of the hard work for you. The hierarchy contains thousands and thousands of lines of correct, optimized and robust Windows code. Many of the member functions that you call invoke code that would have taken you weeks to write yourself. In this way MFC tremendously accelerates your project development cycle.

MFC is fairly large. For example, Version 4.0 of the hierarchy contains something like 200 different classes. Fortunately, you don't need to use all of them in a typical program. In fact, it is possible to create some fairly spectacular software using only ten or so of the different classes available in MFC. The hierarchy is broken into several different class categories which include:

- Application Architecture
- Graphical Drawing and Drawing Objects
- File Services
- Exceptions
- Structures - Lists, Arrays, Maps
- Internet Services
- OLE 2
- Database
- General Purpose

### **CHECK YOUR PROGRESS**

- Write the steps to create a menu.
- How dialog box can be created?
- What is the role of VC++ in MFC?



---

## 3.12 SUMMARY

---

Normally VC++ comes within Microsoft Visual Studio. Visual Studio also contains Visual Basic, Visual C#, and Visual J#. Using Visual Studio, one can mix and match languages within a "solution". VC++ is a programming environment that contains all the libraries, examples, and documentation needed to create applications for Windows. Instead of talking about programs, it is talked about project and solution. Solutions can contain several projects and projects typically contain multiple items or files.

**Object** means a real word entity such as pen, chair, table etc. **Collection of objects** is called a class. It is a logical entity. VC++ is a multi-paradigm programming language. Meaning, it supports different programming styles. One of the popular ways to solve a programming problem is by creating objects, known as object-oriented style of programming.

The *classes* are the most important features of C++ that leads to Object Oriented programming. Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance of that class. The variables inside class definition are called as data members and the functions are called member functions.

Each project usually has one text-format resource script (RC) file that describes the project's menu, dialog, string, and accelerator resources. The RC file also contains *#include* statements to bring in resources from other subdirectories. These resources include project-specific items, such as bitmap (BMP) and icon (ICO) files, and resources common to all VC++ programs, such as error message strings.

The VC++ *debugger* has been steadily improving, but it doesn't actually fix the bugs yet. The debugger works closely with Visual C++ to ensure that breakpoints are saved on disk. *ClassWizard* is a program (implemented as a DLL) that's accessible from VC++'s View menu. ClassWizard takes the drudgery out of maintaining VC++ class code. VC++ 6.0 contains a number of useful diagnostic tools. For example, SPY++ gives a tree view of system's processes, threads, and windows. It also let view messages and examine the windows of running applications.

Apart from framework all other objects in a windows are teated as resources. A separate .rc file holds the description of the resource. Resource compiler which is required to built into VC++ IDE and called automatically when a project has resources included. A *resource* is a text file that allows the compiler to manage objects such as pictures, sounds, mouse cursors, dialog boxes, etc.

An *event* is a message sent by an object within a program to the main program loop, informing it that something has happened. An application is made of various objects. Most of the time, more than one application is running on the computer and the operating system is constantly asked to perform some assignments. *Menus* allow to arrange commands in a logical and easy-to-find fashion. With the Menu editor, menus can be created and edited by working directly with a menu bar that closely resembles the one in the finished application.

Applications for Windows frequently communicate with the user through *dialog boxes*. CDialog class provides an interface for managing dialog boxes. The VC++ dialog editor makes it easy to design dialog boxes and create their dialog-template resources.

Most of the *file processing* in an MFC application is performed in conjunction with a class named CArchive. The CArchive class serves as a relay between the application and the medium used to either store data or make it available. It allows to save a complex network of objects in a permanent binary form (usually disk storage) that persists after the deleted objects.

---

### 3.13 TERMINAL QUESTIONS

---

1. What do you understand by object oriented language? Explain its features.
2. Explain VC++ components briefly.
3. Compare procedure oriented language over object oriented.
4. Explain some characteristics of a class.
5. Write a short note on resources.
6. Describe the term dialog box and its creation in VC++.
7. Discuss the steps to create menus.
8. What do you understand by event? Explain event handling.
9. Write a short note on file handling.



॥ सरस्वती नः सुभगा मयस्करत् ॥

Uttar Pradesh Rajarshi Tandon  
Open University

Bachelor in Computer  
Application

**BCA-118**

**Windows Programming**

**BLOCK**

**2**

**VISUAL BASIC PROGRAMMING**

---

**UNIT-4**

**Windows Programming**

---

**UNIT-5**

**Working with Controls**

---

**UNIT-6**

**Dialog Boxes and Internet**

---

---

## Course Design Committee

---

**Prof. Ashutosh Gupta**

Director

School of Science, UPRTOU Prayagraj

**Prof. Suneeta Agarwal**

Dept. of Computer Science & Engineering

Motilal Nehru National Institute of Technology, Allahabad, Prayagraj

**Dr. Upendra Nath Tripathi**

Associate Professor

Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

**Dr. Ashish Khare**

Associate Professor

Dept. of Computer Science, University of Allahabad, Prayagraj

**Ms. Marisha**

Assistant Professor (Computer Science)

School of Science, UPRTOU Prayagraj

**Mr. Manoj Kumar Balwant**

Assistant Professor (Computer Science)

School of Sciences, UPRTOU Prayagraj

---

## Course Preparation Committee

---

**Dr. Krishan Kumar**

**Author**

Assistant Professor

Department of Computer Science,

Gurukula Kangri Vishwavidyalaya Haridwar (UK)

**Dr. Brajesh Kumar**

**Editor**

Associate Professor, Dept. of CS & IT

M.J.P Rohilkahand University, Bareilly, Uttar Pradesh

**Prof. Ashutosh Gupta**

**Director (In-Charge)**

School of Computer & Information Sciences

UPRTOU Prayagraj

**Mr. Manoj Kumar Balwant**

**Coordinator**

Assistant Professor (computer science)

School of Sciences, UPRTOU Prayagraj

---

©UPRTOU, Prayagraj - 2020

ISBN :

---

©All Rights are reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the **Uttar Pradesh Rajarshi Tondon Open University, Prayagraj.**

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.

**Printed By :** Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

---

## BLOCK INTRODUCTION

---

*Unit 4* describes about the Visual Basic programming concepts. Firstly, it explains the history/evolution of Visual Basic thereafter it deals with the basic building features like variables, data types, decision making, procedures, looping, events, functions, modules etc. It also includes the difference between the C++ and VB. After going through this unit it would be able to make the programs in Visual Basic easily and in less time. This unit focused on Visual Basic 6.0.

*Unit 5* describes about the controls available in control/tool box and other custom controls. It also tells about the creation and use of controls. This unit basically describes about the controls of Visual Basic like Text box, Command Buttons, List boxes, Combo Boxes, Picture box, Image Control, Shape Controls, Timer, Scrollbars, Frames, Checkboxes, Option Boxes, Frames, File, Drive and Directory List boxes, RichTextBox, Tree View Control, List view Control, Progressbar, Menus, Grid Controls. For the creation of all controls normally two ways have been discussed.

*Unit 6* explains about the relation between Visual Basic and Internet. Moreover, it describes about the Modal and Modeless dialog boxes. As we know that dialog boxes are the main components of Windows programming. These modal and modeless are the two important dialog boxes used in event-driven programming. Furthermore, it explains Common Dialog Controls viz Message Dialog Box, Save as dialog box, Font dialog box, File dialog box, Print Dialog box.



---

# UNIT-4 WINDOWS PROGRAMMING

---

## Structure

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Introduction to Visual Basic
- 4.3 Important Windows
- 4.4 Variables
- 4.5 Data Types
- 4.6 Decision Making
- 4.7 Operators
- 4.8 Loops
- 4.9 Procedure in Visual Basic
- 4.10 Visual Basic Code Module
- 4.11 Summary
- 4.12 Terminal Questions

---

## 4.0 INTRODUCTION

---

This unit basically describes the Visual Basic (VB) programming concepts in simple way. **VB**, is a programming **language developed by Microsoft**, is commonly used to develop Windows-based applications. It is event-driven programming language and very easy to use and understand, it provides various tools to create applications and software easily and quickly. It is based on events and hence known as event driven programming language. Event driven means the code is only executed when an event occurs. Almost all applications for Windows are event driven, for example nothing happens in any program in Windows until one clicks a menu or type a text or press enter to submit the command or query like. All actions made by user or any i/o devices are called events.

**The most popular version of VB** is 6.0. It is a good language to make the programming concepts more strong. After reading this unit it could be easy to understand fundamental concepts of VB. It is a popular language for some years. **Still it is very useful** and widely used. **Especially it is a boon for the beginners.** Applications can be easily developed using the VB in very less time and without writing too much code. It provides the basic **programming concepts** and techniques that the most other languages normally provide. VB was derived from **BASIC** (Beginners All-purpose Symbolic Instruction Code). BASIC was

developed in 1960 by Professor **John Kemeny** and **Thomas Kurtz**. The purpose of BASIC was to provide an easy programming language for Windows Applications. Microsoft designed VB in 1991 and released its many versions time to time. But out of all the versions VB 6.0 is very famous due to some reasons that will be discussed in detail in subsequent sections. The latest version is Visual Basic.net or VB.Net.

VB is configured in many editions to make it fit for all types of developers. There are three different type of **versions as follows**.

- 1) **Learning Edition:** Learning edition is released for the **students/learners**. It is equipped with all basic controls and tools to get started.
- 2) **Professional Edition:** This version is for developers who already have programming **experience**. Learning Edition, controls, and wizards are included.

**Enterprise Edition:** Enterprise edition is for the **advanced developers or software personnel**. In this edition, more number of controls, tools and functions are included to perform advance database operations.

---

## 4.1 OBJECTIVES

---

At the end of this unit you will be able to understand the following:

- Integrated Development Environment (IDE)
- Object based language
- Object oriented language
- Data types in VB
- Variables
- Decision making
- Loops
- Functions
- Modules

---

## 4.2 INTRODUCTION TO VISUAL BASIC

---

Visual Basic (VB) is a programming language and development environment created by Microsoft. It is an extension of the *BASIC* programming language that combines BASIC functions and commands with visual controls. Visual Basic provides a graphical user interface GUI that allows the developer to drag and drop objects into the program as well as manually write program code. It, also referred to as "VB," is designed to make software development easy and efficient, while still being powerful enough to create advanced programs. For example, the Visual Basic language is designed to be "human readable," which means the source code can be understood without requiring lots of comments. The



VB program also includes features like "IntelliSense" and "Code Snippets," which automatically generate code for visual objects added by the programmer. Another feature, called "AutoCorrect," can debug the code while the program is running.

Programs created with VB can be designed to run on Windows, on the Web, within Office applications, or on mobile devices. Visual Studio, the most comprehensive VB development environment, or IDE, can be used to create programs for all these mediums. Visual Studio .NET provides development tools to create programs based on the .NET framework, such as ASP.NET applications, which are often deployed on the Web. Finally, Visual Basic is available as a streamlined application that is used primarily by beginning developers and for educational purposes.

---

## 4.2.1 HISTORY

---

Visual Basic (VB) is a High Level Language having Object Oriented RAD (Rapid Application Development) environment for Windows operating system. The First version of VB was written for Windows 3.0, but it was largely accepted till version 3.1 of Windows. VB programming is Event-Driven, where user selects different objects according to their use in common interface known as Form. The Programmer can generate different actions associated with objects to trigger codes to perform some action as call of procedures, and functions, etc.

VB has several advancements according to computational growth. In 1991 VB 1.0 was introduced for Windows, which had features of connecting GUI to programming. It was originally developed by Alan Cooper and his associative developers for Windows. In September 1992, VB 1.0 for DOS was released, which was also compatible with Windows. In November 1992, VB 2.0 was introduced that provided better programming environment with better performance. In 1993, VB 3.0 was released, which had functionality of database connectivity with MS Access database. In August 1995, VB 4.0 was introduced, which had features of writing classes. In February 1997, VB 5.0 was released for 32-bit Windows operating system. The programs written in VB 5.0 were interoperable with version 4.0 and vice versa. In 1998, VB 6.0 was released having features of web development supporting Internet Explorer. The web based applications could be developed with this version.

---

## 4.2.2 CHARACTERISTICS OF VB

---

**GUI:** Visual Basic is a GUI based language. This means that a Visual Basic program always shows something on the screen that the user can interact with to get a job done.

**Modularization:** The Visual Basic supports concept of modules. It is considered a good programming practice to modularize your programs. Small modules where it is clearly indicated what comes into the module and what goes out makes a program easy to understand.

**Object Oriented Features:** Object oriented is a concept, where the programmer thinks of the program in terms of "objects" that also interact with each other. The Visual Basic supports these features and forces this good programming practice.

**Debugging:** The Visual Basic offers two different options for code debugging i.e. 1) Debugging Managed Code Runtime Debugger and 2) The Debugging Managed Code. These options debug C, C++ applications and Visual Basic Windows applications. The Runtime Debugger in Visual Basic helps to find and fix bugs in programs at runtime.

**Data Access Feature:** By using data access features, we can create databases, scalable server-side components for most databases, including Microsoft SQL Server and other enterprise-level databases.

**Macros IDE:** The Macros integrated development environment is similar in design and function to the Visual Studio IDE. The Macros IDE includes a code editor, tool windows, properties windows, and editors.

---

### 4.2.3 OBJECT BASED PROGRAMMING

---

Object based programming languages follow all principles same as object oriented programming languages have except inheritance and polymorphism. In VB programming environment every component such as Button, Text box, etc., which is visible to programmer or user is an object. The methods, actions, and properties are associated with these objects and hence can be used for application development. But object based languages need not to support inheritance and polymorphism, because there is constraint on objects that they cannot invoke actions on other objects. It has the following features:

1. Object-based language doesn't support all features of OOPs like polymorphism and inheritance.
2. Object-based language has in-built objects like JavaScript has window object.
3. VB is an object based language because you can use class and its objects but cannot inherit one class into another class i.e. it does not support inheritance.
4. Moreover, functions cannot be overloaded/overridden i.e. it does not support polymorphism.
5. Except Visual Basic, another example of object-based language is JavaScript.

---

### 4.2.4 BASIC CONCEPTS

---

The Visual Basic programming interface is also known as Integrated Development Environment (IDE) because it provides the tools for creating, interpreting and generating executable files on one interface. On starting the VB editor, IDE displays a dialog box as shown in Fig. 4.1. Different options such as start new project, open existing project, and list of recently used/opened projects appear. If you click on the *new project* option, several options such as Standard EXE, Active EXE, Active DLL, ActiveX Control appear to develop different types of applications as per the requirement. To develop an application, which is executable program, Standard EXE should be selected.



**Fig. 4.1 VB 6.0 Dialog Box**

Following applications can be created using VB IDE:

**Standard Exe:** A Standard EXE project is a typical application.

**ActiveX EXE, ActiveX DLL:** These options are available only with professional version of VB. These are OLE automation servers, additional functionality can be added using these applications.

**ActiveX Control:** It is the part of professional edition of VB. It provides control over different objects such as TextBox or Command Button, which is a basic element of user interface. Moreover, own controls can also be added if they are not present in ActiveX Control.

**ActiveX Document EXE, ActiveX Document DLL:** ActiveX documents are essential for those VB applications, which support hyper linking (as in case of Web Browsers.)

**Application Wizard, Wizard Manager:** Wizard is a collection of windows that collect information from user, when user fills up all the information, the wizard proceeds to make an application as per user requirement.

**Data Project:** This is identical to Standard Exe project but has Database Connectivity feature, adds control to the ToolBox for data retrieval, manipulation; and also used with ActiveX Control for report generation.

**DHTML Application:** This is used to create Dynamic HTML pages; these can be used to display browser's window on client computer.

**IIS Application:** It can be used to create applications which can run on the web server, and communicate with client over local host or over the Internet.

**AddIn:** These are additional commands that can be used in Application.

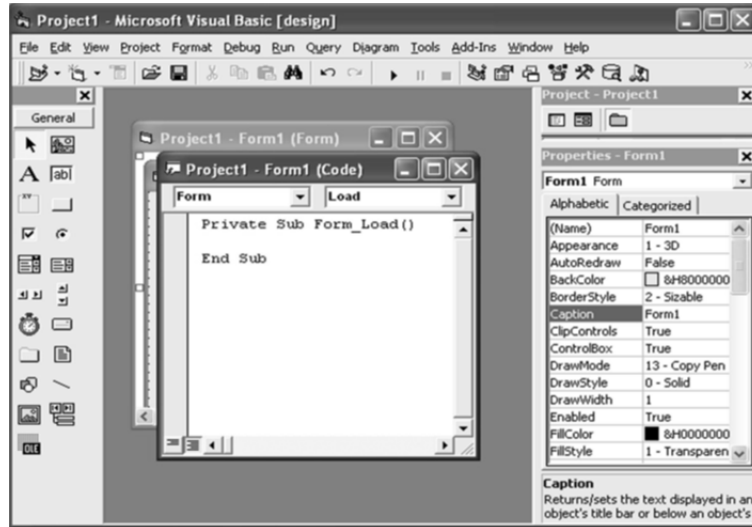
**VB Enterprise Edition Controls:** It creates Standard EXE project and loads all tools which are with the package of VB 6.0.

---

## 4.2.5 CREATING FIRST VISUAL BASIC APPLICATION

---

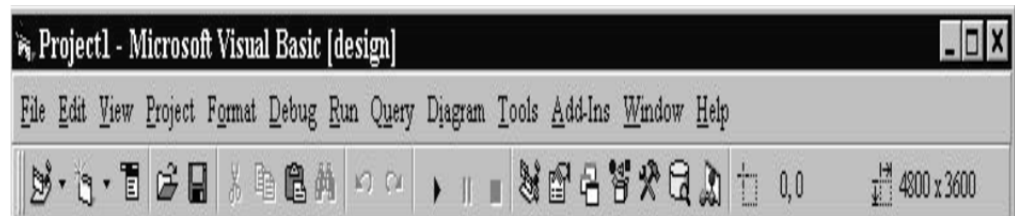
When we select Standard EXE to develop an application project window appears as shown in Fig. 4.2.



**Fig. 4.2 Project Window**

The default page contains the name as Form1, we can change the Project name as well as Form name if we require. There are different parts of this form, by double clicking we get code view of the form where different objects can be used according to our need, the source code window having list of objects and procedures listed with them. The Different parts of opening window are listed below:

- Title Bar
- Menu Bar
- Tool Bar



**Fig. 4.3 Title Bar, Menu Bar, and Tool Bar**

**Title Bar:** Title Bar shows the current project name, VB operating mode and current form.

**Menu Bar:** Menu Bar is next to the Title Bar and contains the various menus like file menu, edit menu, and project menu, etc. The options available in the menu bar are described below:

1. *File* –It is used for all file operations like opening and saving projects, creating exe files, and listing of recent projects.
2. *Edit* –It contains options such as copy, paste, undo, find, and replace, etc. for easy editing.
3. *View* –Used for showing and hiding different components in the user interface.
4. *Project* –It contains commands for adding components in the current project.
5. *Format* –It can be used for aligning the controls on the form.
6. *Debug* –Used for debugging options in the current running project.
7. *Run* –It deals with commands for executing the current project, starting a project, breaking the project, and ending operations of the project.
8. *Query* –It is used for database operations based on SQL.
9. *Diagram*–It contains commands for editing diagrams of database, used for building database applications.
10. *Tools* –It consist of tools for creating ActiveX controls and components.
11. *Add-Ins* –Users can add and remove Add-Ins that are available with VB IDE using Add-In Manager.
12. *Window* –It is used for arranging and managing windows.
13. *Help* –Used for helping options supported by VB Developer.

**Tool Bar:** This is located next to the Menu Bar and contains different icons for different tasks such as creating a new project, adding a form, opening a file, saving a project, and running and stopping the project, etc. It also provides shortcuts to commonly used menus. There are different kinds of tools present in VB IDE, some of them are listed below:

1. Edit property
2. Debug property
3. Form Editor Toolbar
4. Style property
5. Add and remove methods
6. Customizing toolbar

---

## 4.3 IMPORTANT WINDOWS

---

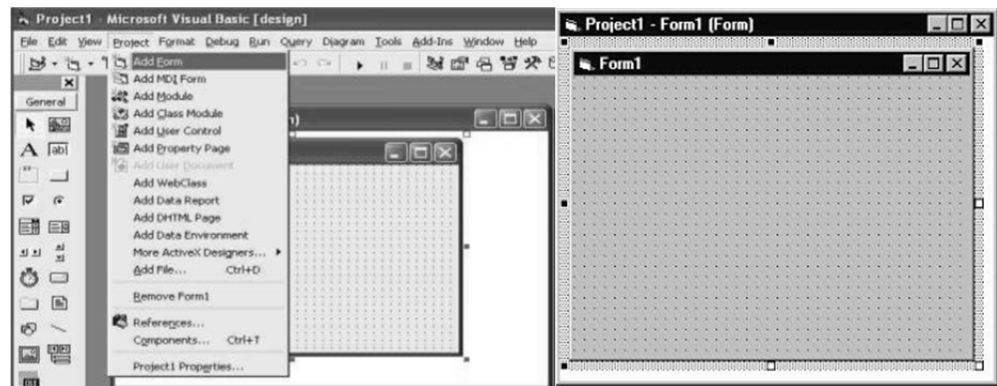
An important part of Visual Basic is the ability to create Windows Forms applications that run locally on users' computers. You can use Visual Studio to create the application and user interface using Windows Forms. A Windows Forms application is built on classes from the *System.Windows.Forms* namespace.

---

### 4.3.1 FORM WINDOW

---

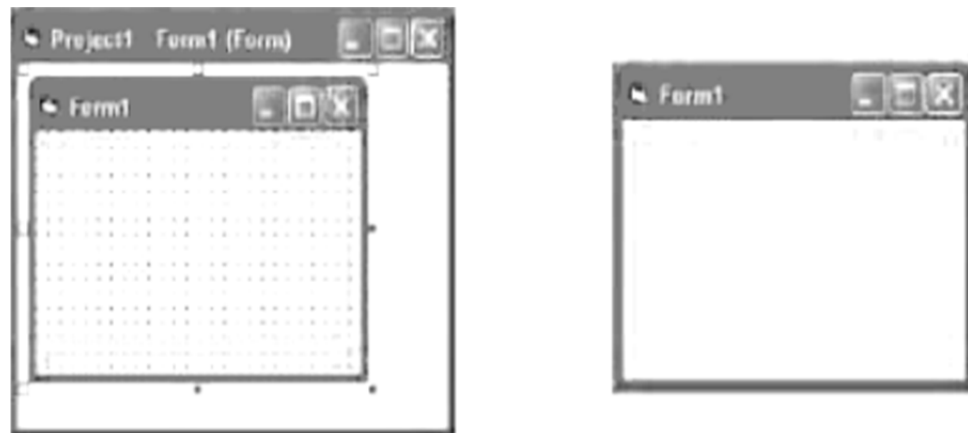
When you start VB project (Fig. 4.4), a default form (Form1) with a standard grid (a window consisting of regularly spaced dots) appears in a pane called the Form window. You can use the Form window grid to create the user interface and to line up interface elements. The form is most important object for application perspective. The form holds other objects that provide users/programmers ease to use different computing ability present in VB. In the standard project the form designer creates and modify the form. The designer can add many forms as per the application requirement. The addition of forms can be done from project menu.



**Fig. 4.4 Adding the Form to Project (left), a new form (right)**

The form has two modes as shown in Fig. 4.5:

- *The Design Mode*—This mode contains the environment where designer can add different objects to the form as per as requirement.
- *The Running Mode*—The running mode is the view when the application is executed.



**Fig. 4.5 Design View (left), Running View (right)**

---

### 4.3.1.1 BUILDING INTERFACE ELEMENTS

---

To build the interface elements, you click an interface control in the VB toolbox, and then you draw the user interface element on your form by using the mouse. This process is usually a simple matter of clicking to position one corner of the element and then dragging to create a rectangle the size you want. After you create an element, suppose a text box, you can refine it by setting properties for the element. In a text box, for example, you can set properties to make the text boldface, italic, or underlined. You can adjust the size of the form by using the mouse — the form can take up part or the entire screen. To control the placement of the form when you run the program, adjust the placement of the form in the Form Layout window.

---

### 4.3.1.2 FORM PROPERTIES

---

In VB, every object has properties associated with it. A form is an important container as it contains other objects within it. It has different properties, which have default values that can be modified according to the requirement of user. The changes can be made either by rewriting or by using down arrow on side. Few of them can be rewritten or browsing computer files when the designer clicks on dotted button on the right side of property selected. The important properties are shown in Table 4.1.

**Table 4.1** Properties of Form Window

Property	Objective	Code	State of Changing
Caption	For Naming the form	Form1.caption="User_Defined_Name"	Design and Run
BackColor	for setting Background color	Form1.BackColor="color_name"	Design and Run
ForeColor	for Setting Foreground color	Form1.ForeColor="color_name"	Design and Run
Enabled	To enable or disable tools	Form1.Enable="True or False"	Design and Run
Hide	To hide the form	Form1.hide	Run
Show	To show the form	Form_no.show	Run

---

### 4.3.1.3 EVENTS

---

The designer put all his declarations and executable statements within the procedure or events defined in the code window. Since VB is an event-driven programming language, where all the objects are associated with events and produce effects according to event generated by object. Some important events supported by form are shown in Table 4.2.

**Table 4.2** Events of Form

Event	Action taken (Event-Driven on)
Click	Single click on Form
DbClick	Double click on Form
Load	Lading the Form

**Example:**

```
Private Sub Form_Load ()  
  
Form1.show  
  
Print "Welcome to Visual Basic"  
  
End Sub
```

---

### 4.3.2 PROPERTY WINDOW

---

With the Properties window (Fig. 4.6), the characteristics of the user interface elements on a form can be changed. It is known as property settings. A property setting is a characteristic of a user interface object. For example, you can change the text displayed by a text box control to a different font, point size, or alignment. With VB, you can display the text in any font installed on your system, just as it is done in Microsoft Excel or Microsoft Word.



**Fig. 4.6** Property Window



To display the Properties window, click the Properties Window button on the toolbar. If the window is currently docked, you can enlarge it by double-clicking the title bar. To redock the Properties window, double-click its title bar again.

The Properties window contains the following elements:

- A drop-down list box at the top of the window, from which you select the object whose properties you want to view or set.
- Two tabs, which list the properties either alphabetically or by category.
- A description pane that shows the name of the selected property and a short description of it.

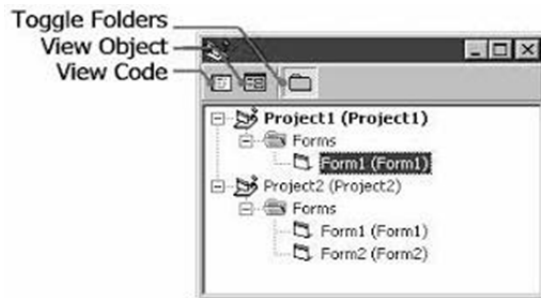
You can change property settings by using the Properties window while you design the user interface or by using program code to make changes while the program runs.

---

### 4.3.3 PROJECT WINDOW

---

A VB program consists of several files that are linked together to run the program. The VB 6.0 development environment includes a Project window to help you switch back and forth between these components as you work on a project as shown in Fig. 4.7.



**Fig. 4.7 Project Window**

The Project window lists all the files used in the programming process and provides access to them with two special buttons: *View Code* and *View Object*. To display the Project window, click the Project Explorer button on the VB toolbar. If the window is currently docked, you can enlarge it by double-clicking the title bar. To re-dock the Project window, double-click its title bar again.

The project file maintains a list of all the supporting files in a VB programming project. You can recognize project files by their .vbp file name extensions. You can add individual files to and remove them from a project by using commands on the Project menu. The changes that you make will be reflected in the Project window. If you load additional projects into Visual Basic with the File menu's Add Project command, outlining symbols appear in the Project window to help you organize and switch between projects.

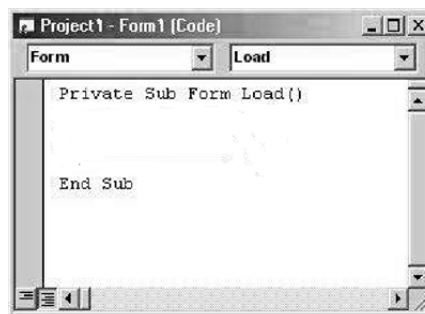
**Note:-** In VB versions 1 through 3, project files had the .mak file name extension. In Visual Basic versions 4, 5, and 6.0, project files have the .vbp file name extension.

---

### 4.3.4 CODE WINDOW

---

You can create a major portion of your program by using controls and setting properties. However, the most VB programs require some additional program code that acts as the brain behind the user interface that you create. This computing logic is created using program statements, keywords, identifiers, and arguments that clearly spell out what the program should do each step during the execution. As soon as one enters in program statements of the Code window, a special text editing window designed specifically for VB program code appears. The Code window can be displayed in either by clicking View Code in the Project window or by clicking the View Menu Code command (Fig. 4.8).



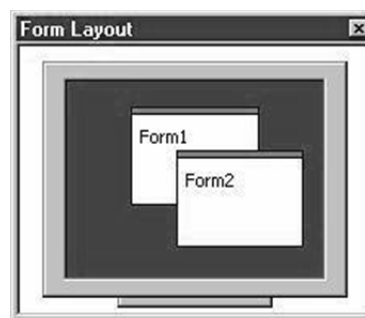
**Fig. 4.8 The Code window**

---

### 4.3.5 FORM LAYOUT WINDOW

---

The Form Layout window is a visual design tool (Fig. 4.9). With it, you can control the placement of the forms in the Windows environment when they are executed. When you have more than one form in your program, the Form Layout window is especially useful. You can arrange the forms onscreen exactly the way you want. To position a form in the Form Layout window, simply drag the miniature form to the desired location in the window.



**Fig. 4.9 Form Layout Window**

---

## 4.4 VARIABLES

---

Variables are the data names, which are used to store some value defined by user in a program. The variables are declared and defined by the programmer according to need during application development. These variables are used to assign some value associated with the object. On approaching some event the values associated with object is going to transfer to the variables. Following rules are used for *naming convention of variables* in Visual Basic:

1. The first character must be a letter.
2. The variable name may include character, digit, underscore.
3. No more than 40 characters can be included in naming.
4. No use of reserved words is allowed.

The variable declaration has three different forms:

1. Default declaration
2. Implicit declaration
3. Explicit declaration

---

### 4.4.1 DEFAULT DECLARATION

---

If the variables in the program of VB are not declared either implicitly or explicitly, then the declaration would be done automatically and it would be a default variant value. It is a special type that can have numeric, string, or date data i.e. any data type which depends on the type of data being used.

---

### 4.4.2 IMPLICIT DECLARATION

---

If you set **Option Explicit** to **Off**, you can implicitly declare a variable by simply using it in your code. The compiler assigns the **Object Data Type** to all implicitly declared variables. However, your application is more efficient if you declare all your variables explicitly and with a specific data type. This reduces the incidence of naming-conflict errors and spelling mistakes. It also lets the compiler detect potential run-time errors such as assigning an **Integer** to a **Short**.

You can write a procedure in which you do not declare a local variable. The following example illustrates this.

```
Function safeSqrt(num)
    ' Make sure num is positive for square root.
    tempVal = Math.Abs(num)
    Return Math.Sqrt(tempVal)
```

End Function

Visual Basic automatically creates temp Val as a local variable, which you can use as if you had declared it explicitly. While this is convenient, it can lead to subtle errors in your code if you misspell a variable name. Suppose you had written the procedure in the preceding example as follows:

```
Function safeSqrt(num)

' Make sure num is positive for square root.

    tempVal = Math.Abs(num)

    Return Math.Sqrt(temVal)

End Function
```

At first glance, this code looks the same. But because the tempVal variable is misspelled as the argument to Sqrt, the compiler creates an additional local variable called temVal, which is never assigned a value, and your function always returns zero.

---

### 4.4.3 EXPLICIT DECLARATION

---

Had explicit declaration been in effect for the source file containing the safeSqrt procedure in the preceding example, Visual Basic would have recognized tempVal and temVal as undeclared variables and generated errors for both of them. As a result, you would then explicitly declare tempVal. The following example illustrates this.

```
Function safeSqrt(ByVal num As Double) As Double

' Make sure num is positive for square root.

    Dim tempVal As Double = Math.Abs(num)

    Return Math.Sqrt(temVal)

End Function
```

With this revised code, you would understand the problem immediately because Visual Basic would display an error message for the incorrectly spelled temVal. Because explicit declaration helps you catch these kinds of errors, it is recommended that you use it with all your code.

Hence, Explicit declaration means it is done by programmer, and the values are predefined hence the variables are called explicit variables.

***Example:***

```
Dim number1 As Integer
Dim number2 As Integer
Dim Name As String
```

### **CHECK YOUR PROGRESS**

- How a variable can be defined in VB?
- Compare implicit and explicit declaration.
- Name some crucial properties of Form window.

---

## **4.5 DATA TYPES**

---

Usually, we come across all kinds of data in our daily life. For example, we need to handle data such as names, addresses, money, date, stock quotes, statistics, and many more everyday. Similarly, in programming, we have to deal with all sorts of data. Some data elements can be processed mathematically, while some other data elements are in text format or other forms. VB divides data into different types so that they are easier to manage when we need to write the code involving those data. VB 6.0 classifies the data elements into two major data types: numeric data types and the non-numeric data types.

---

### **4.5.1 NUMERIC DATA TYPE**

---

Numeric data types represent those data elements that can be processed mathematically with standard operators e.g. height, weight, share values, the price of goods, monthly bills, and fees etc. In VB, numeric data are divided into 7 categories, depending on the range of values they can store. Calculations that only involve round figures can use Integer or Long integer in the computation. Programs that require high precision calculation need to use Single and Double decision data types, they are also called floating point numbers. For currency calculation, currency data types can be used. Lastly, if even more precision is required to perform calculations that involve many decimal points, we can use the decimal data types. These data types summarized in detail in table 4.3.

**Table 4.3 Numeric Data Types**

<b>Type</b>	<b>Storage</b>	<b>Range of Values</b>
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.

Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

---

## 4.5.2 NON-NUMERIC DATA TYPE

---

Non-numeric data types are used for those data elements that cannot be manipulated mathematically. Non-numeric data comprise strings, date, and boolean, etc. that store only two values (true or false), object data type and Variant data type. They are summarized in detail in table 4.4.

**Table 4.4 Non-numeric Data Types**

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

---

## 4.6 DECISION MAKING

---

**Decision making** structures require that the **programmer** specifies one or

more conditions to be evaluated or tested by the **program**, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false. Moreover, the decision making is used to control the flow of program execution. VB supports control structures such as *if... Then*, *if...Then ...Else*, and *Select...Case*.

---

#### **4.6.1 IF...THEN SELECTION STRUCTURE**

---

In VB, the *If...Then* selection structure performs an indicated action only when the condition is True; otherwise the action is skipped. It can be understood using following syntax.

**Syntax:**

```
If <condition> Then
    statement
End If
e.g.: If average>75 Then
    txtGrade.Text = "A"
End If
```

---

#### **4.6.2 IF...THEN...ELSE SELECTION STRUCTURE**

---

The *If...Then...Else* selection structure allows the programmer to specify that a different action is to be performed when the condition is True in comparison when the condition is False.

**Syntax:**

```
If <condition > Then
    statements
Else
    statements
End If
e.g.: If average>50 Then
    txtGrade.Text = "Pass"
Else
    txtGrade.Text = "Fail"
End If
```

---

#### **4.6.3 NESTED IF...THEN...ELSE SELECTION STRUCTURE**

---

Nested *If...Then...Else* selection structures test for multiple cases by placing *If...Then...Else* selection structures inside *If...Then...Else* structures.

Syntax of the Nested If... Then... Else selection structure. You can use Nested If either of the methods as shown above

**Method 1:**

```
If < condition 1 > Then
    statements
ElseIf < condition 2 > Then
    statements
ElseIf < condition 3 > Then
    statements
Else
    Statements
End If
```

**Method 2:**

```
If < condition 1 > Then
    statements
Else
    If < condition 2 > Then
        statements
    Else
        If < condition 3 > Then
            statements
        Else
            Statements
        End If
    End If
EndIf
```

**Example:** To find the grade using nested if and display in a text box

```
If average > 75 Then
    txtGrade.Text = "A"
ElseIf average > 65 Then
    txtGrade.Text = "B"
ElseIf average > 55 Then
    txtGrade.text = "C"
ElseIf average > 45 Then
    txtGrade.Text = "S"
```



```
Else  
txtGrade.Text = "F"  
End If
```

---

#### 4.6.4 *SELECT...CASE* SELECTION STRUCTURE

---

*Select...Case* structure is an alternative to *If...Then...ElseIf* for selectively executing a single block of statements from among multiple block of statements. *Select...case* is more convenient to use than the *If...Else...End If*. The following program block illustrate the working of *Select...Case*.

**Syntax:**

```
Select Case Index  
Case 0  
Statements  
Case 1  
Statements  
End Select
```

**Example:** Find the grade using *Select...Case* and display in the appropriate text box

```
Dim average as Integer  
average = txtAverage.Text  
Select Case average  
Case 100 To 75  
txtGrade.Text ="A"  
Case 74 To 65  
txtGrade.Text ="B"  
Case 64 To 55  
txtGrade.Text ="C"  
Case 54 To 45  
txtGrade.Text ="S"  
Case 44 To 0  
txtGrade.Text ="F"  
Case Else  
MsgBox "Invalid average marks"  
End Select
```

---

## 4.7 OPERATORS

---

Simply, an operator operates between two operands e.g.  $x = y + z$ . Here addition is the operator and  $y$  and  $z$  are the two operands. Operators operate all the function using the variables or operands. In VB 6.0, there are three types of operators:

- 1) *Arithmetic operators*- Perform familiar calculations on numeric values, including shifting their bit patterns.
- 2) *Relational operators*- compare two expressions and return a Boolean value representing the result of the comparison
- 3) *Logical operators*- combine Boolean or numeric values and return a result of the same data type as the values

---

### 4.7.1 ARITHMETIC OPERATORS

---

In Visual Basic, *Arithmetic Operators* are useful to perform the basic arithmetic calculations like addition, subtraction, division, etc. based on user's requirements. The following table lists the different types of arithmetic operators available in Visual Basic.

**Table 4.5 Arithmetic Operators**

Operators	Description	Example	Result
+	Addition	5+5	10
-	Subtraction	10-5	5
/	Division	25/5	5
\	Integer Division	20\3	6
*	Multiplication	5*4	20
^	Exponent (power of)	3^3	27
Mod	Remainder of division	20 Mod 6	2
&	String concatenation	"ATI"& &"Kurunegala"	"ATI Kurunegala"

---

### 4.7.2 RELATIONAL OPERATORS

---

Relational operators are used to compare values. These operators always result in a boolean value. Relational operators in VB like greater than, less than, equal to etc. play important role in conditional statements. It is to be noted that in VB, the comparison operator equal to is not like in C and C influenced languages. Moreover, the relational operators are not limited to numbers. We can use them

for other objects as well. Although they might not always be meaningful. Detail of these are shown below in table 4.6.

**Table 4.6 Relational Operators**

<b>Operators</b>	<b>Description</b>	<b>Example</b>	<b>Result</b>
>	Greater than	10 > 8	True
<	Less than	10 < 8	False
>=	Greater than or equal to	20 >= 10	True
<=	Less than or equal to	10 <= 20	True
<>	Not Equal to	5 <> 4	True
=	Equal to	5 = 7	False

---

### 4.7.3 LOGICAL OPERATORS

---

Logical operators allow you to evaluate one or more expressions and return a Boolean value (True or False). In Visual Basic, we have the following logical operators (Table 4.7). Boolean operators are also called logical. Boolean operators are used to work with truth values. To understand a logical operator, we construct a table to list its possible inputs and outputs. This table is known as truth table.

**Table 4.7 Logical Operators**

<b>Operators</b>	<b>Description</b>
OR	Operation will be true if either of the operands is true
NOT	Operation will be true if not equal of the operands is true
AND	Operation will be true only if both the operands are true

---

### 4.7.4 ORDER OF OPERATIONS

---

The evaluation of expressions in Visual Basic is ordered by its operator precedence as shown below in Table 4.8.

**Table 4.8 Priority of Operators**

Priority	Name of Operator	Notation Used
11 (Highest)	Exponentiation	^
10	Unary identity and negation	(unary)+ (unary)-
9	Multiplication and floating-point division	* /
8	Integer division	\
7	Modulus arithmetic	Mod
6	Addition and subtraction	+ -
5	String concatenation	&
4	Relational/comparison operators	= <> < <= > >=
3	Negation	Not
2	Conjunction	And
1 (Lowest)	Inclusive disjunction	Or

**CHECK YOUR PROGRESS**

- Describe any two data types.
- What is decision making?
- How many operators does VB support?

---

## 4.8 LOOPS

---

In computer programming, a *loop* is a sequence of instructions that is continuously repeated until a certain condition is reached. Typically, a certain process is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number or not. A repetition structure allows the programmer to repeat an action until given condition is false. There are different types of loops in VB, which differ in their structure and way of functioning.

---

### 4.8.1 DO WHILE... LOOP STATEMENT

---

The *Do While...Loop* is used to execute statements until a certain condition is met. The following syntax for *Do While... Loop* counts from 1 to 100.

```
Dim number As Integer
number = 1
Do While number <= 100
    number = number + 1
Loop
```

A variable *number* is initialized to 1 and then the *Do While... Loop* starts. Firstly, the condition is tested; if condition is True, then the statements are executed. When it gets to the Loop it goes back to the Do and tests condition again. If condition is False on the first pass, the statements are never executed.

---

### 4.8.2 WHILE... WEND STATEMENT

---

A *while...Wend* statement behaves like the *Do While...Loop* statement. The following *While...Wend* counts from 1 to 100:

```
Dim number As Integer
number = 1
While number <= 100
    number = number + 1
Wend
```

---

### 4.8.3 DO ... LOOP WHILE STATEMENT

---

The *Do...Loop While* statement first executes the statements and then tests the condition after each execution. The following program block illustrates the structure:

```
Dim number As Long
number = 0
```

```
Do
  number = number + 1
Loop While number < 201
```

The above code executes the statements between *Do* and *Loop While* structure at least once, and then it determines whether the counter is less than 201. If so, the program again executes the statements between *Do* and *Loop While*, else exits the Loop.

---

#### **4.8.4 DO UNTIL...LOOP STATEMENT**

---

Unlike *Do While...Loop* and *While...Wend* repetition structures, the *Do Until...Loop* structure tests a condition for falsity. Statements in the body of a *Do Until...Loop* are executed repeatedly as long as the loop-continuation test evaluates to False. An example for *Do Until...Loop* statement. The coding is typed inside the click event of the command button

```
Dim number As Long
number=0
Do Until number > 1000
  number = number + 1
Print number
Loop
```

Numbers between 1 to 1000 will be displayed on the form as soon as you click on the command button.

---

#### **4.8.5 FOR...NEXT LOOP**

---

The *For...Next* Loop is another way to make loops in Visual Basic. *For...Next* repetition structure handles all the details of counter-controlled repetition. The following loop counts the numbers from 1 to 100:

```
Dim x As Integer
For x = 1 To 50
  Print x
Next
```

In order to count the numbers from 1 to 50 in steps of 2, the following loop can be used

```
For x = 1 To 50 Step 2
  Print x
Next
```

The code given above displays the numbers vertically. Another loop given

as follows counts the numbers alternatively as 1, 3, 5, 7 etc. and displays them horizontally

```
For x = 1 To 50
  Print x & Space$ (2);
Next
```

To increase the difference between the numbers, increase the value inside the brackets after Space\$. Another example of *For...Next* repetition structure is given as follows. It also uses the *If* conditional statement.

```
Dim number As Integer
For number = 1 To 10
  If number = 4 Then
    Print "This is number 4"
  Else
    Print number
  End If
Next
```

The output is "This is number 4" instead of number 4.

---

## 4.9 PROCEDURES IN VISUAL BASIC

---

VB offers different types of procedures to execute small sections of code in applications. VB programs can be broken into smaller logical components called procedures. Procedures are useful for condensing repeated operations such as the frequently used calculations, text, and control manipulations, etc. The benefits of using procedures in programs are:

- It is easier to debug a program with procedures, which breaks a program into discrete logical limits.
- Procedures used in one program can act as building blocks for other programs with slight modifications.
- A procedure can be sub function or property procedure.

---

### 4.9.1 SUB PROCEDURES

---

A sub procedure can be placed in standard, class, and form modules. It is created using keywords *Sub* and *End Sub*. Each time the procedure is called, the statements between keywords i.e. *Sub* and *End Sub* are executed. The syntax for a sub procedure is as follows:

```
[Private | Public] [Static] Sub Procedurename [(arglist)]
  [statements]
```

End Sub

Where *arglist* is a list of argument names separated by commas. Each argument acts like a variable in the procedure. There are two types of sub procedures namely general procedures and event procedures.

---

## 4.9.2 EVENT PROCEDURES

---

An event procedure is a procedure block that contains the actual name of control, an underscore (\_), and the event name. The following syntax represents the event procedure for a Form\_Load event.

```
Private Sub Form_Load()  
    ....statement block..  
End Sub
```

Event procedures acquire the declarations as Private by default.

---

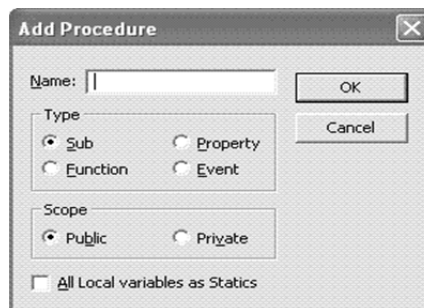
## 4.9.3 GENERAL PROCEDURES

---

A general procedure is declared when several event procedures perform the same actions. It is a good programming practice to write common statements in a separate procedure (general procedure) and then call them in the event procedure.

In order to add General procedure:

- The *Code Window* is opened for the module to which the procedure is to be added.
- The *Add Procedure* option is chosen from the *Tools* menu, which opens an *Add Procedure* dialog box as shown in the Fig 4.10 given below.
- The name of the procedure is typed in the *Name* text box.
- Under *Type*, *Sub* is selected to create a Sub procedure, *Function* is selected to create a Function procedure or *Property* to create a Property procedure as shown in Fig. 4.10.
- Under *Scope*, *Public* is selected to create a procedure that can be invoked outside the module, or *Private* to create a procedure that can be invoked only from within the module as shown in Fig. 4.10.



**Fig. 4.10 Adding Procedure**



We can also create a new procedure in the current module by typing Sub procedure name, Function procedure name, or Property procedure name in the Code window. A Function procedure returns a value while Sub procedure does not return a value.

---

## 4.9.4 FUNCTION PROCEDURES

---

Functions are like sub procedures, except they return a value to the calling procedure. They are especially useful for taking one or more pieces of data, called *arguments* and performing some tasks with them. The function returns a value that indicates the results of the tasks completed by the function. The following Function procedure calculates the third side or hypotenuse of a right triangle, where A and B are the other two sides. It takes A and B as two arguments (of data type Double) and returns the third side.

```
Function Hypotenuse (A As Double, B As Double) As Double
```

```
Hypotenuse = sqr (A^2 + B^2)
```

```
End Function
```

The above function procedure is written in the general declarations section of the Code window. A function can also be written by selecting the *Add Procedure* dialog box from the Tools menu and by choosing the required scope and type. In Function, the function Name is followed by argument field. Arguments are important part as function call is initiated by passing some type of data. The arguments either could be data elements or reference to memory address. Therefore, there are two methods to pass arguments to a function,

- Passing arguments by reference
- Passing arguments by Value

---

### 4.9.4.1 PASSING ARGUMENT BY REFERENCE

---

Passing arguments by reference provides the function to access the actual variables. The calling function passes the addresses of data elements to the called function. This approach does not require additional copying memory. Any changes made to the arguments are reflected back to the memory location. Therefore, if any instruction belonging to the same function/program or any other program accesses those memory locations, it gets the modified values.

*For Example:*

```
Function Add(var1 As integer, var2 As Integer) As Integer
```

```
Add = var1+var2
```

```
var1=0
```

```
var2=0
```

```
End Function
```

If above function is called using the following code,

```
Dim A As Integer, B As Integer
A=20
B=2
Sum=Add(A,B)
Debug.Print A
Debug.Print B
Debug.Print Sum
```

The output would be

```
20
2
22
```

---

#### 4.9.4.2 PASSING ARGUMENTS BY VALUE

---

When argument is passed as value, the argument value is copied to its local variables and function performs its operation on the local variables. Any changes made to the argument values are not reflected to the original memory location. It is completely transparent outside the function body. Therefore, the function needs to explicitly return the resultant value if required.

For Example:

```
Function Degree (ByVal Celsius as Single) As Single
Degree=(9/5)*celsius +32
End Function
```

Here the *ByVal* keyword is used to ensure that the function is always called by value.

---

#### 4.9.5 PROPERTY PROCEDURES

---

A property procedure is a series of Visual Basic statements that manipulate a custom property on a module, class, or structure. Property procedures are also known as *property accessors*. A property procedure is used to create and manipulate custom properties. It is used to create read only properties for Forms, Standard modules and Class modules.

Visual Basic provides for the following property procedures:

- A Get procedure returns the value of a property. It is called when you access the property in an expression.

- A Set procedure sets a property to a value, including an object reference. It is called when you assign a value to the property.

Usually property procedures can be defined in pairs, using the Get and Set statements, but you can define either procedure alone if the property is read-only (Get Statement) or write-only (Set Statement). You can omit the Get and Set procedure when using an auto-implemented property. Moreover, you can define properties in classes, structures, and modules. Properties are Public by default, which means you can call them from anywhere in your application that can access the property's container.

---

#### 4.9.5.1 DECLARATION SYNTAX

---

A property itself is defined by a block of code enclosed within the Property Statement and the End Property statement. Inside this block, each property procedure appears as an internal block enclosed within a declaration statement (Get or Set) and the matching End declaration.

---

#### 4.9.5.2 DATA TYPE

---

A property's data type and principal access level are defined in the Property statement, not in the property procedures. A property can have only one data type. For example, you cannot define a property to store a Decimal value but retrieve a Double value.

---

#### 4.9.5.3 ACCESS LEVEL

---

However, you can define a principal access level for a property and further restrict the access level in one of its property procedures. For example, you can define a Public property and then define a Private Set procedure. The Get procedure remains Public. You can change the access level in only one of a property's procedures, and you can only make it more restrictive than the principal access level.

---

### 4.10 VISUAL BASIC CODE MODULE

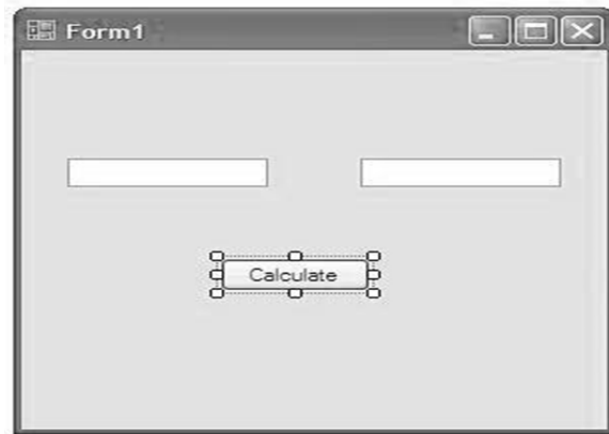
---

VB application source code is structured into *module* files with a *.vb* suffix. By default, Visual Studio creates a separate module file for each form in an application containing the code to construct the form. For example, the code to create a form called *Form1* will be placed in a module file named *Form1.Designer.vb*. Similarly, any code that has been defined by the developer to handle events from controls in the form will be placed by Visual Studio into a module file called *Form1.vb*.

When writing additional code for an application, the code should ideally be logically grouped together with other source code in a module file. Logical grouping means the code should be grouped with other code of a similar nature. For example, code to work with files might be placed in a module called *FileIO.vb*, while mathematical procedures might all reside in a file named *Math.vb*. The idea is to ensure VB code must be placed in a file where it makes

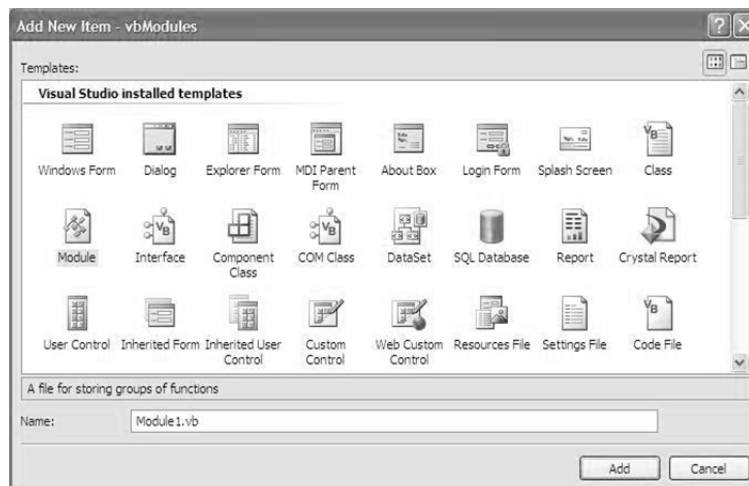
sense for it to be located.

As mentioned previously, the Visual Studio places the code to construct each form in separate module files. Now it is needed to learn how to create a new module in a project to contain our own VB code. Beginning by creating a new Windows Application project in Visual Studio called *vbModules*, two TextBox controls (named *value1TextBox* and *value2TextBox*) and a button (labeled *Calculate*) to the Form have been added as shown in Fig. 4.11.



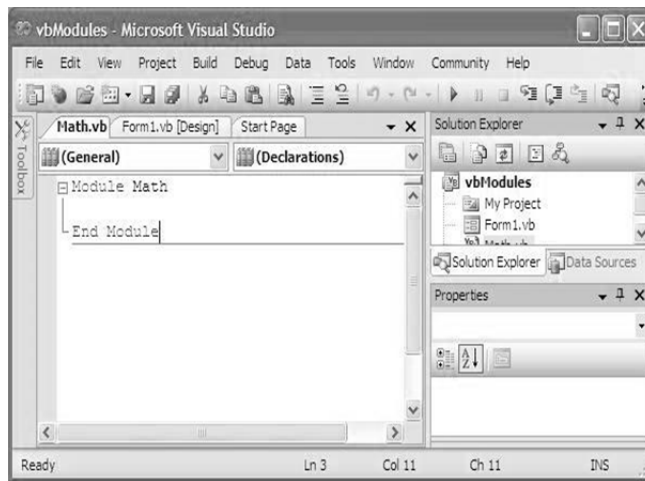
**Fig. 4.11 Simple Form containing Text Box and Button**

Once the new project has been opened and the first form is visible, select *Add Module...* from the *Project* menu. The *Add Item* window will appear (Fig. 4.12) with the *Module* item pre-selected.



**Fig. 4.12 Adding Module**

Name the new module *Math.vb* and click the *Add* button. The new module will be added to the project and a new tab labeled *Math.vb* for accessing the module code will appear (Fig. 4.13) in the design area.



**Fig. 4.13 Code Window for Module**

### **CHECK YOUR PROGRESS**

- Define *while...wend* statement.
- What do you mean by procedure?
- Write the use of code module.

---

## **4.11 SUMMARY**

---

*Variables* are the data names which are used to store some value defined by user in a program. They are used to assign some value associated with the objects. On approaching some event the values associated with the objects are transferred to the variables. In VB, the variables can be declared in three ways-implicit declarations, explicit declarations, and default declarations.

*Numeric data types* are used for the variables that are supposed to consist of numbers for processing mathematically with standard operators such as height, weight, share values, the price of goods, monthly bills, and fees, etc. In VB, numeric data types are divided into 7 types, depending on the range of values they can store. Calculations that only involve round figures can use Integer or Long integer in the computation. Programs that require high precision calculation need to use Single and Double decision data types.

*Nonnumeric data types* are the data types that cannot be manipulated mathematically. These data types comprise string data types, date data types, boolean data types, object data type and variant data type.

*Decision making* structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false. It is used to control the flow of program's execution. Visual Basic supports control structures such as *if... Then, if...Then ...Else, Select...Case*.

Similar to many other languages, VB has three types of operators– 1) Arithmetic operators e.g. addition, subtraction, division, multiplication; 2) Relational operators e.g. greater than, greater than or equal to, less than, less than or equal to etc.; 3) Logical operators e.g. OR, NOT, AND.

The *Do While...Loop* is used to execute statements until a certain condition is met. First, the condition is tested; if condition is True, then the statements are executed. When it gets to the Loop it goes back to the Do and tests condition again. If condition is False on the first pass, the statements are never executed. It is called entry control loop. The *Do...Loop While* statement first executes the statements and then test the condition after each execution. It is called exit control loop. The *For...Next* Loop is another way to code loops in programming of Visual Basic. *For...Next* repetition structure handles all the details of counter-controlled repetition.

Unlike the *Do While...Loop* and *While...Wend* repetition structures, the *Do Until... Loop* structure tests a condition for falsity. Statements in the body of a *Do Until...Loop* are executed repeatedly as long as the loop-continuation test evaluates to False.

*Functions* are like sub procedures, except they return a value to the calling procedure. They are especially useful for taking one or more pieces of data, called arguments and performing some tasks with them. Then the functions return a value that indicates the results of the tasks complete within the function.

---

## 4.12 TERMINAL QUESTIONS

---

1. Discuss the evolution of VB programming.
2. Explain important characteristics of VB.
3. Write a short note on Properties window.
4. Explain explicit and implicit declaration in VB.
5. Compare procedure oriented language with object oriented language.
6. Explain numeric data types.
7. Write a program in VB to reverse an entered number.
8. Write a program for the addition of two numbers using function.
9. Discuss the concept of procedures in detail.
10. Define the term loop. Explain types of loop statements in VB.
11. Write a short note on module.

---

# UNIT-5 WORKING WITH CONTROLS

---

## Structure

- 5.0 Introduction
- 5.1 Objectives
- 5.2 What is Control
- 5.3 What is Custom Control
- 5.4 Control Properties
- 5.5 The Intrinsic Controls
- 5.6 Rich Text Box Controls
- 5.7 Working with Menu Items
- 5.8 Adding and Removing Control
- 5.9 Naming a Control
- 5.10 Summary
- 5.11 Terminal Questions

---

## 5.0 INTRODUCTION

---

There are so many command controls in Visual Basic to work with it. Using these controls, programming becomes easy for programmers. These controls are: command buttons, text boxes, check boxes, picture box, image controls, labels, lists, scroll bars, combo box, RichTextbox, progress bars, menu, web browser etc. Some commonly used command controls are buttons, text fields, and labels. VB is different from other languages as it contains various command controls and event procedures accordingly.

VB toolbox contains the controls which are used in the form design. The controls are used by users to interact with the applications. The controls are categorized as Intrinsic controls, ActiveX controls and Insertable objects. The built-in VB controls are called as intrinsic controls; these controls are always available in the toolbox. The intrinsic controls are– Text box, Check box, Combo box, Command button, Dirlist, Drive list, Frame, Horizontal scrollbar, Vertical scrollbar, Image control, Label, List box, Option button, Timer etc.

All the controls which exist as separate files with an extension .ocx are called ActiveX controls. These controls are available in all versions of the professional and enterprise VB editions. The ActiveX controls are- Common

Dialog, DataCombo Box, DataList, MSFlexGrid, ADO Data Control, Animation Control, Communication control, CoolBar Control, etc. In VB, it is possible to include other application objects in a program and automate it, the applications are inserted as objects.

All controls have various properties which could be either set from the property window or in code window. Before coding an event procedure for the control to respond to an event, you have to set certain properties for the control to determine its appearance and actions with the event procedure. You can set the properties of the controls in the properties window or at runtime. Every control works on a particular event like mouse up, mouse down, key up, key down, click, double click, form load unload etc.

---

## 5.1 OBJECTIVES

---

At the end of this unit you will come to know about the following:

- Discussing the various controls
- Adding the controls on the form
- Updating the properties of control
- Command buttons
- Text box
- Combo box
- List box
- Labels
- Progress bars
- Scroll bars

---

## 5.2 WHAT IS A CONTROL

---

A control is a special type of object that a programmer draws on a Form to enable user interaction with an application. In VB, the most of the objects appear inside forms. All objects that appear inside a form are called controls. Menus and menu items are all controls. The tools that appear in the Toolbox are also a kind of controls and can be placed on a form by double-clicking or by click-drag operation onto a form. Moreover, controls have some names by default e.g. the default name of a form window is Form1. Controls, normally perform two things, either they accept user input or display output. Controls have properties that define attributes of their appearance, such as caption, position, size, and color; and also aspects of their behavior, such as how they respond to user input. Controls can respond to events initiated by the user. They also respond to the events triggered by the system. For example, if one wants that whenever the command button is selected it should perform the action desired and should display the result. For this you can write code in a Command Button's click event



procedure that would load a file or perform a calculation and then display the result.

In addition to properties and events, one can use methods to manipulate controls from within the code. For example, one can use the Move method with some controls to change their location and size. Each control has its own set of properties, events, and methods. Graphical controls include the Image, Label, Line, and Shape controls. A graphical control uses fewer system resources and has different drawing and display characteristics than other controls.

---

## 5.3 WHAT IS A CUSTOM CONTROL

---

The custom control is a program that someone has written which can be included in the VB program. The two types of custom controls that VB can use are VBX (Visual Basic custom extension controls) and OCX (OLE Custom extension controls). A file with a .VBX or .OCX extension or an insertable object that when added to a project using the custom controls dialog box, extends the toolbox. The custom controls are nothing but also programs. They are also likely to have the bugs in them. This might create a problem since the source code of these controls is not available with you and without that it is not possible to remove the bug.

Custom Controls dialog box can be used to load custom controls and insertable objects to the project's toolbox. To open the Custom Controls dialog box, choose Custom Controls from the Tools menu. This operation can also be performed by pressing Ctrl + T in VB IDE. The dialog box displays the available custom controls and insertable objects. You also find the box appearing on the left of the list of controls. If the insertable objects or controls are not appearing completely, you can click at the respective boxes and then the complete list appears. Either or both of the options can be selected using the following options:

- Insertable Objects option displays insertable objects, such as a Microsoft Excel Chart.
- Controls option displays controls with .OCX and .VBX filename extensions.
- Selected Items Only option displays only those items in the Available Controls which you have been selected to include in the project.

If you want to load a Custom control each time VB is loaded, use the following procedure:

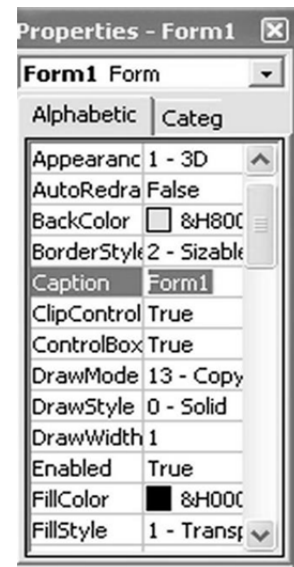
- To add a control or insertable object to the Toolbox, click on the check box next to its name. You find 'X' mark appearing in the box.
- To remove a control or insertable object from the project, click the check box next to its name. Clicking again will deselect the control from the list available; and 'X' mark will disappear from the check box.
- When you are finished making selections, click on OK button to update your Toolbox.

---

## 5.4 CONTROL PROPERTIES

---

In properties window, the currently selected object appears at the top. At the bottom part, the items listed in the left column represent the names of various properties associated with the selected object, while the items listed in the right column represent the states of the properties. Properties can be set by highlighting the items in the right column then change them by typing or selecting the options available. For example, in order to change the caption, just highlight Form1 under the name Caption and change it to other names. You may also alter the appearance of the form by setting it to 3D or flat, change its foreground and background color, change the font type and font size, enable or disable, minimize and maximize buttons and more. You can also change the properties at runtime to give special effects such as change of color, shape, animation effect and so on.



**Fig. 5.1** Property Window

Example 5.1 shows the code that will change the form color to red every time the form is loaded. VB uses the hexadecimal system to represent the color. You can check the color codes in the properties windows which are showing up under *ForeColor* and *BackColor*.

**Example 5.1:** Program to change background color. This example changes the background colour of the form using the *BackColor* property.

```
Private Sub Form_Load()  
  
Form1.Show  
  
Form1.BackColor = &H000000FF&  
  
End Sub
```

**Example 5.2:** Program to change shape. This example is to change the control's Shape using the *Shape* property. This code will change the shape to a circle at runtime.

```
Private Sub Form_Load()  
  
Shape1.Shape = 3  
  
End Sub
```

A few important points about setting up the properties are:

- You should set the Caption Property of a control clearly so that a user knows what to do with that command.

- Use a meaningful name for the Name Property because it is easier to write and read the event procedure and easier to debug or modify the programs later.
- One more important property is whether to make the control enabled or not.
- Finally, you must also consider making the control visible or invisible at runtime, or when should it become visible or invisible.

**CHECK YOUR PROGRESS**

- Define the term control in Visual Basic programming.
- What is a Custom control and how it can be added?
- Give the name of the properties associated with the form control.

---

## 5.5 THE INTRINSIC CONTROLS

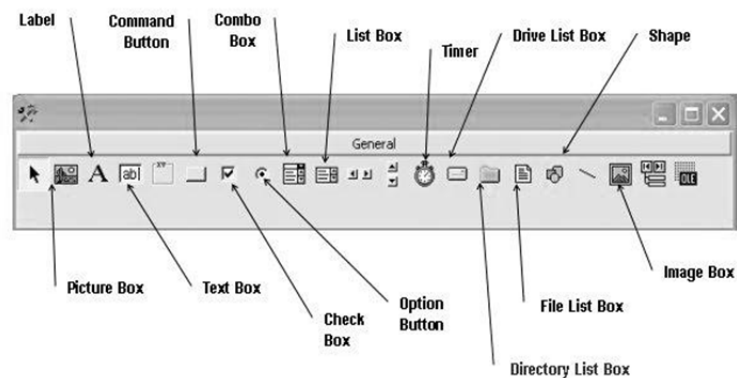
---

The intrinsic controls are available automatically whenever VB program is created. During design time, you can access the basic controls can be accessed (see Fig 5.2) from the VB6 Toolbox. Table 5.1 lists the intrinsic controls.

**Table 5.1 The VB6 Intrinsic Controls**

<i>Control</i>	<i>Description</i>
Label	Displays text on a form
Frame	Serves as a <i>container</i> for other controls
CheckBox	Enables users to select or deselect an option
ComboBox	Allows users to select from a list of items or add a new value
HscrollBar	Allows users to scroll horizontally through a list of data in another control
Timer	Lets your program perform actions in real time, without user interaction
DirListBox	Enables users to select a directory or folder
Shape	Displays a shape on a form
Image	Displays graphics (images) on a form but can't be a container

OLE Container	Enables you to add the functionality of another Control program to your program
PictureBox	Displays graphics (images) on a form and can serve as a container
TextBox	Can be used to display text but also enables users to enter or edit new or existing text
CommandButton	Enables users to initiate actions
OptionButton	Lets users select one choice from a group; must be used in groups of two or more
ListBox	Enables users to select from a list of items
VscrollBar	Enables users to scroll vertically through a list of data in another control
DriveListBox	Lets users select a disk drive
FileListBox	Lets users select a file
Line	Displays a line on a form
Data	Lets your program connect to a database



**Fig. 5.2 Tool Box**

### **5.5.1 The TextBox**

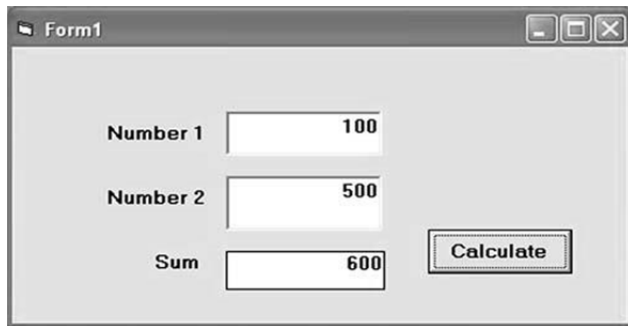
The TextBox control is the standard control for accepting input from the user as well as to display the output. It can handle string (text) and numeric data

but not images or pictures. A string entered into a text box can be converted to a numeric data by using the function Val(text). The following example illustrates a simple program that processes the input from the user.

**Example 5.3:** In this program, two text boxes are inserted into the form together with a few labels. The two text boxes are used to accept inputs from the user and one of the labels will be used to display the sum of two numbers that are entered into the two text boxes. Besides, a Command button is also programmed to calculate the sum of the two numbers using the plus operator. The program creates a variable sum to accept the summation of values from TextBox1 and TextBox2. The procedure to calculate and to display the output on the label is given below.

```
Private Sub Command1_Click()  
    'To add the values in TextBox1 and TextBox2  
    Sum = Val(Text1.Text) + Val(Text2.Text)  
    'To display the answer on label 1  
    Label1.Caption = Sum  
End Sub
```

The output is shown in Fig. 5.3



**Fig. 5.3 TextBox**

---

### 5.5.2 THE LABEL

---

This is probably the first control you will master. It is used to display static text, titles and screen output from operations. The label is a very useful control for Visual Basic programming, as it is not only used to provide instructions and guides to the users, it can also be used to display outputs. One of its most important properties is *Caption*. Using the syntax *Label.Caption*, it can display text and numeric data. You can change its caption either in the properties window or at runtime. Use of label has been shown in previous Example 5.3 and Fig. 5.3 which was used to show the working of TextBox.

---

### 5.5.3 THE COMMANDBUTTON

---

The Command Button is one of the most important controls as it is used to

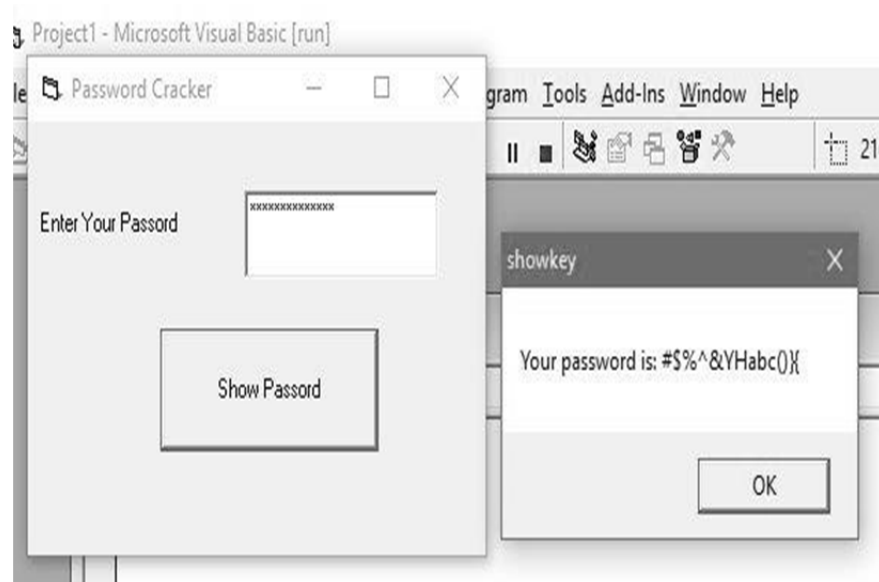
execute commands. It displays an illusion that the button is pressed when the user clicks on it. The most common event associated with the command button is the click event, and the syntax for the procedure is

```
Private Sub Command1_Click ()  
    Statements  
End Sub
```

**Example 5.4** A Simple Password Cracker: *In this program, a secret password entered by the user is to be cracked. In the design phase, insert a command button and change its name to cmd\_ShowPass. Next, insert a TextBox and rename it as TxtPassword and delete Text1 from the Text property. Besides that, set its PasswordChr to \*. Now, enter the following code in the code window*

```
Private Sub cmd_ShowPass_Click()  
    Dim yourpassword As String  
    yourpassword = Txt_Password.Text  
    MsgBox ("Your password is: " & yourpassword)  
End Sub
```

Run the program and enter a password, then click on the Show Password button to reveal the password, as shown in Fig. 5.4



**Fig. 5.4 Password Cracker**

---

### **5.5.4 THE PICTUREBOX**

---

The *Picture Box* is one of the important controls that is used to handle pictures in graphics. You can load a picture at design phase by clicking on the

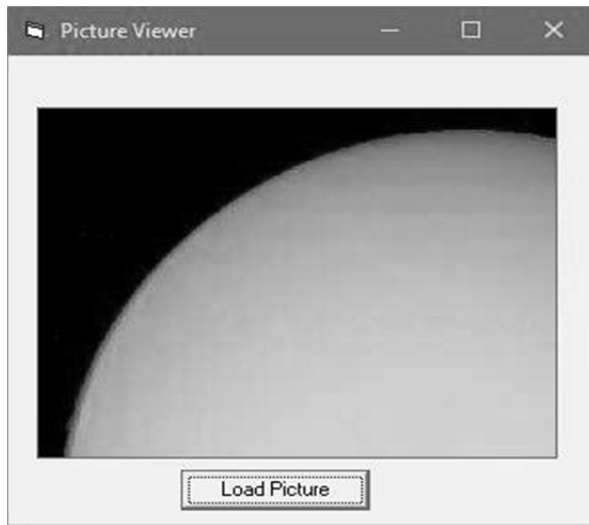
picture item in the properties window and select the picture from the selected folder. You can also load the picture at runtime using the *LoadPicture* method. For example, the statement will load the picture “apple.gif” into the picture box.

```
Picture1.Picture=LoadPicture ("C:\VBprogram\Images\apple.gif")
```

**Example 5.5 Loading Picture:** *In this program, insert a command button and a picture box. Enter the code given below:*

```
Private Sub cmd_LoadPic_Click()  
MyPicture.Picture = LoadPicture("C:\Users\images\planet.jpg")  
End Sub
```

The output is shown in Fig. 5.5.



**Fig. 5.5 Picture Viewer**

---

### 5.5.5 THE IMAGE CONTROL

---

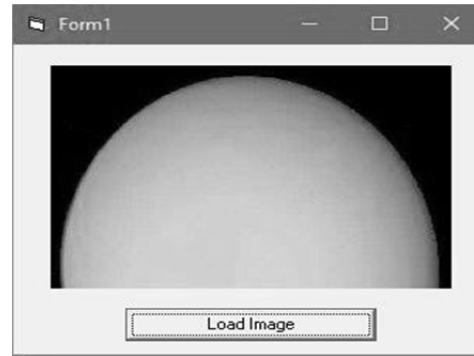
The Image control is another control that handles images and pictures. It functions almost identically to the PictureBox. However, there is one major difference, the image in an Image control is stretchable, which means it can be resized. This feature is not available in the PictureBox control. Similar to the PictureBox, it can also use the LoadPicture method to load the picture. For example, the statement loads the picture applet.gif into the image box.

```
Image1.Picture=LoadPicture ("C:\VBprogram\Images\grape.gif")
```

**Example 5.6 Loading Image:** *In this program, we insert a command button and an Image control into the form. Besides that, we set the image Stretch property to true. Next, enter to following code:*

```
Private Sub cmd_LoadImg_Click()  
MyImage.Picture = LoadPicture("C:\Users\images\planet.jpg")
```

End Sub



**Fig. 5.6 The Image Control**

---

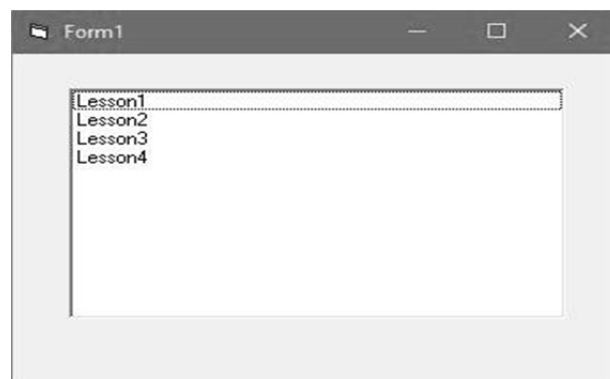
### 5.5.6 THE LISTBOX

---

The function of the ListBox control is to present a list of items where the user can click and select the items from the list. In order to add items to the list, we can use the *AddItem* method. For example, if you wish to add a number of items to List1, following example 5.7 can be used.

*Example 5.7*

```
Private Sub Form_Load()  
    List1.AddItem "Lesson1"  
    List1.AddItem "Lesson2"  
    List1.AddItem "Lesson3"  
    List1.AddItem "Lesson4"
```



**Fig. 5.7 The ListBox**

The items in the ListBox can be identified by the *ListIndex* property, the value of the ListIndex for the first item is 0, the second item has a ListIndex 1, and the third item has a ListIndex 2 and so on.



---

### 5.5.7 THE COMBOBOX

---

The function of the ComboBox control is also to present a list of items where the user can click and select the items from the list. However, the user needs to click on the small arrowhead on the right of the ComboBox to see the items which are presented in a drop-down list. In order to add items to the list, you can also use the *AddItem* method. For example, if you wish to add a number of items to Combo1, you can key in the following statements

*Example 5.8*

```
Private Sub Form_Load()  
    Combo1.AddItem "Item1"  
    Combo1.AddItem "Item2"  
    Combo1.AddItem "Item3"  
    Combo1.AddItem "Item4"  
End Sub
```



**Fig. 5.8 The ComboBox**

---

### 5.5.8 THE CHECKBOX

---

The CheckBox control lets the user select or unselect an option. When the CheckBox is checked, its value is set to 1 and when it is unchecked, the value is set to 0. You can include the statements *Check1.Value = 1* to mark the check box and *Check1.Value = 0* to unmark the check box, as well as use them to initiate certain actions. For example, the program in Example 5.9 will show which items are selected in a message box.

*Example 5.9*

```
Private Sub Cmd_OK_Click()  
    If Check1.Value = 1 And Check2.Value = 0 And Check3.Value = 0 Then  
        MsgBox "Apple is selected"  
    ElseIf Check2.Value = 1 And Check1.Value = 0 And Check3.Value = 0 Then
```

```

MsgBox "Orange is selected"

ElseIf Check3.Value = 1 And Check1.Value = 0 And Check2.Value = 0
Then
MsgBox "Orange is selected"

ElseIf Check2.Value = 1 And Check1.Value = 1 And Check3.Value = 0 Then
MsgBox "Apple and Orange are selected"

ElseIf Check3.Value = 1 And Check1.Value = 1 And Check2.Value = 0 Then
MsgBox "Apple and Pear are selected"

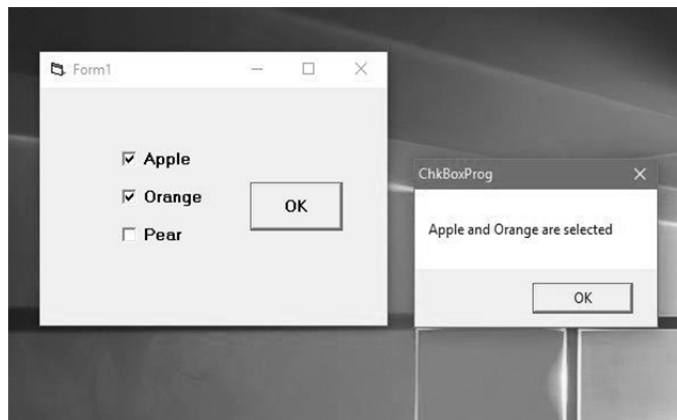
ElseIf Check2.Value = 1 And Check3.Value = 1 And Check1.Value = 0 Then
MsgBox "Orange and Pear are selected"

Else
MsgBox "All are selected"

End If

End Sub

```



**Fig. 5.9 The CheckBox**

---

### **5.5.9 THE OPTIONBUTTON**

---

The *OptionButton* control lets the user select one of the choices. However, two or more option buttons must work together because as one of the option button is selected, the other option button will be unselected. In fact, only one option button can be selected at a time. When an *OptionButton* control is selected, its value is set to “True”; and when it is unselected, its value is set to “False”.

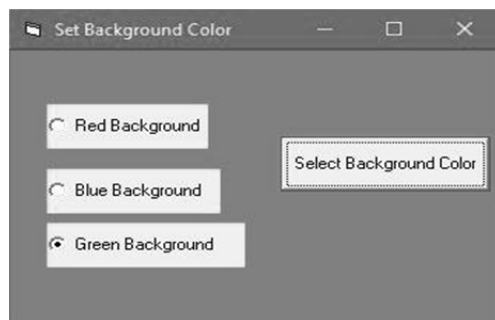
**Example 5.10:** *In this example, we want to change the background color of the form according to the selected option. We insert three option buttons and change their captions to "Red Background", "Blue Background" and "Green Background" respectively. Next, insert a command button and change its name to cmd\_SetColor and its caption to "Set Background Color". Now, click on the command button and enter the following code in the code window:*

```

Private Sub cmd_SetColor_Click()
    If Option1.Value = True Then
        Form1.BackColor = vbRed
    ElseIf Option2.Value = True Then
        Form1.BackColor = vbBlue
    Else
        Form1.BackColor = vbGreen
    End If
End Sub

```

Run the program, select an option and click the "Set Background Color" produces the output, as shown below.



**Fig. 5.10 The OptionButton**

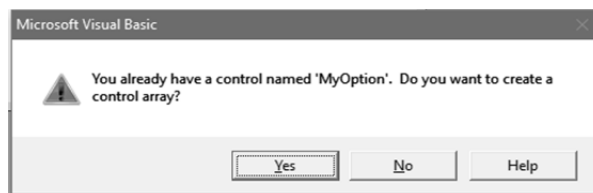
---

### 5.5.10 THE SHAPE CONTROL

---

In the following example, the shape control is placed in the form together with six OptionButtons. To determine the shape of the shape control, we use the shape property. The property values of the shape control are 0, 1, 2, 3, 4, 5 which will make it appear as a rectangle, a square, an oval, a circle, a rounded rectangle, and a rounded square respectively.

***Example 5.11:** In this example, we insert six option buttons. It is better to make the option buttons into a control array as they perform similar action, i.e. to change shape. In order to create a control array, click on the first option button, rename it as MyOption. Next, click on the option button and select copy then paste. After clicking the paste button, a popup dialog (Fig 5.11) box appears asking "Do you want to create a control array?", select yes. The control array can be accessed via its index value, MyOption Index. In addition, we also insert a shape control.*

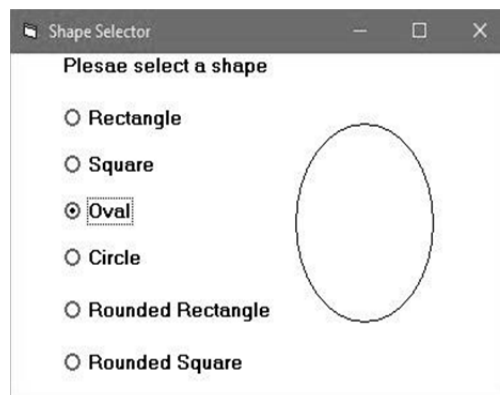


**Fig. 5.11 Dialog Box for Control Array**

Now, enter the code in the code window. We use the *If..Then..Else* program structure to determine which option button is selected by the user.

```
Private Sub MyOption_Click(Index As Integer)
    If Index = 0 Then
        MyShape.Shape = 0
    ElseIf Index = 1 Then
        MyShape.Shape = 1
    ElseIf Index = 2 Then
        MyShape.Shape = 2
    ElseIf Index = 3 Then
        MyShape.Shape = 3
    ElseIf Index = 4 Then
        MyShape.Shape = 4
    ElseIf Index = 5 Then
        MyShape.Shape = 5
    End If
End Sub
```

Run the program and you can change the shape of the shape control by clicking one of the option buttons. The output is shown below.



**Fig. 5.12 The Shape Selector**

**Note:** - A control array is a group of related controls in a Visual Basic form that share the same event handlers. Control arrays are always single-dimensional arrays, and controls can be added or deleted from control arrays at runtime.

---

### 5.5.11 THE DRIVELISTBOX

---

The *DriveListBox* is for displaying a list of drives available in your computer. When you place this control into the form and run the program, you will be able to select different drives from your computer as shown in Fig 5.13.

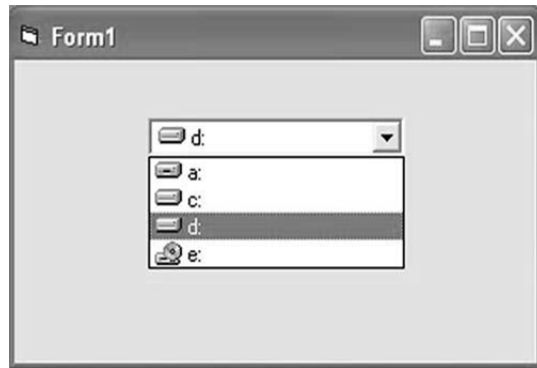


Fig. 5.13 The DriveListBox

---

### 5.5.12 THE DRIVELISTBOX

---

The DirListBox means the Directory List Box. It is for displaying a list of directories or folders in a selected drive. When you place this control into the form and run the program, you will be able to select different directories from a selected drive in your computer as shown below in Fig 5.14.

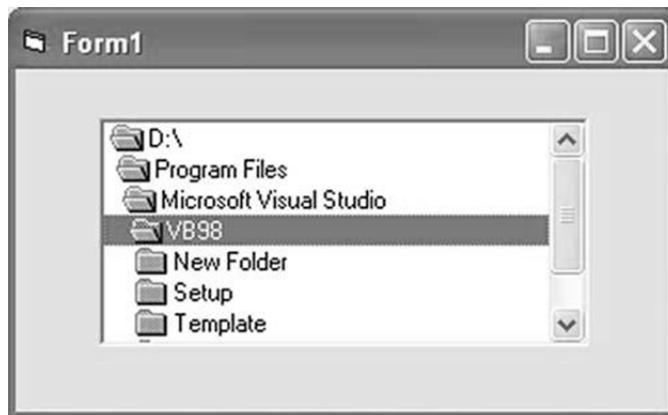


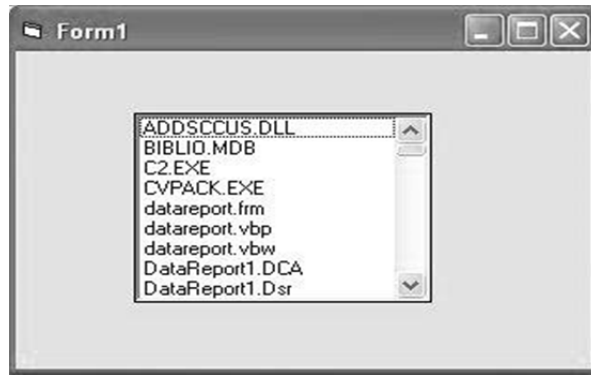
Fig. 5.14 The DirListBox

---

### 5.5.13 THE FILELISTBOX

---

The FileListBox means the File List Box. It is for displaying a list of directories or folders in a selected drive. When you place this control into the form and run the program, you will be able to select different files from a selected drive in the computer as shown below.



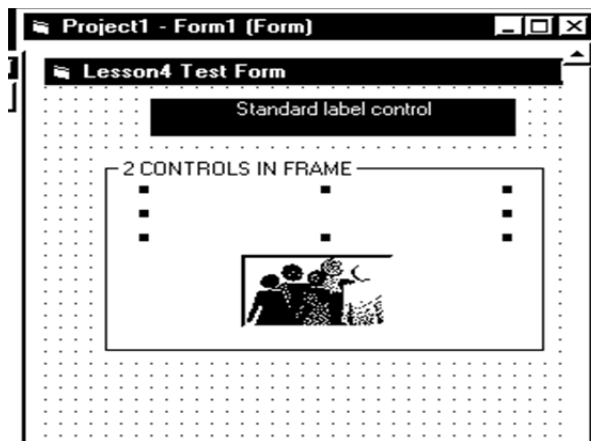
**Fig. 5.15 The FileListBox**

---

### 5.5.14 THE FRAME

---

When you want to group several controls together - name and address, for example - you use a Frame. The colors of form and frame background can be same and only the frame borders are obvious. The colors may be different and clearly stand out. You create the frame before the controls. When you create controls in a frame, they are tied to the frame and move with it. The frame caption is the text that appears at the top of the frame- you use it to define the group.



**Fig. 5.16 The Frame**

---

### 5.5.15 THE TIMER

---

A Timer control is invisible at run time, and its purpose is to send a periodic pulse to the current application. You can trap this pulse by writing code in the Timer's Timer event procedure and take advantage of it to execute a task in the background or to monitor a user's actions. This control exposes only two meaningful properties: Interval and Enabled. Interval stands for the number of milliseconds between subsequent pulses (Timer events), while Enabled lets you activate or deactivate events. When you place the Timer control on a form, its Interval is 0, which means no events. Therefore, remember to set this property to a suitable value in the Properties window or in the *Form\_Load* event procedure.

**Syntax:**

```
Private Sub Form_Load()  
    Timer1.Interval = 500 ' Fire two Timer events per second.  
End Sub
```

Timer controls let you write interesting programs with just a few lines of code. The typical example is a digital clock.

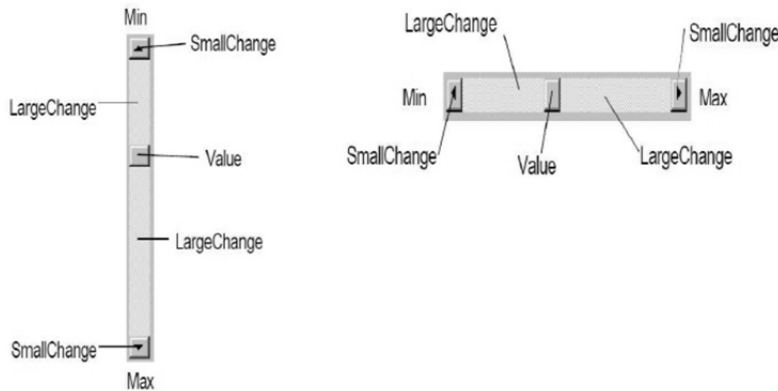
---

### 5.5.16 SCROLL BARS

---

There are two types of scroll bars i.e. horizontal and vertical. These are widely used in windows applications. These provide an intuitive way to move through a list of information and make great input devices. Some properties of scroll bars are mentioned below.

- **LargeChange**– Increment added to or subtracted from the scroll bar value property when the bar area is clicked.
- **Max**– the value of the horizontal scroll bars at the far right and the value of the vertical scroll bar at the bottom can range from -32,768 to 32,767.
- **Min**- The horizontal scroll bar at the left and the vertical scroll bar at the top can range from -32,768 to 32,767.



**Fig. 5.17 The Scrollbars**

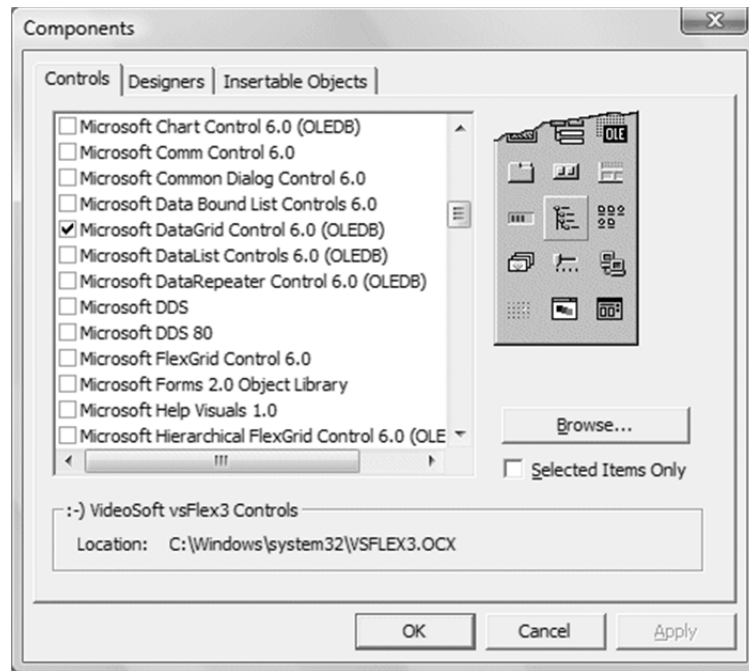
---

### 5.5.17 DATAGRID CONTROL

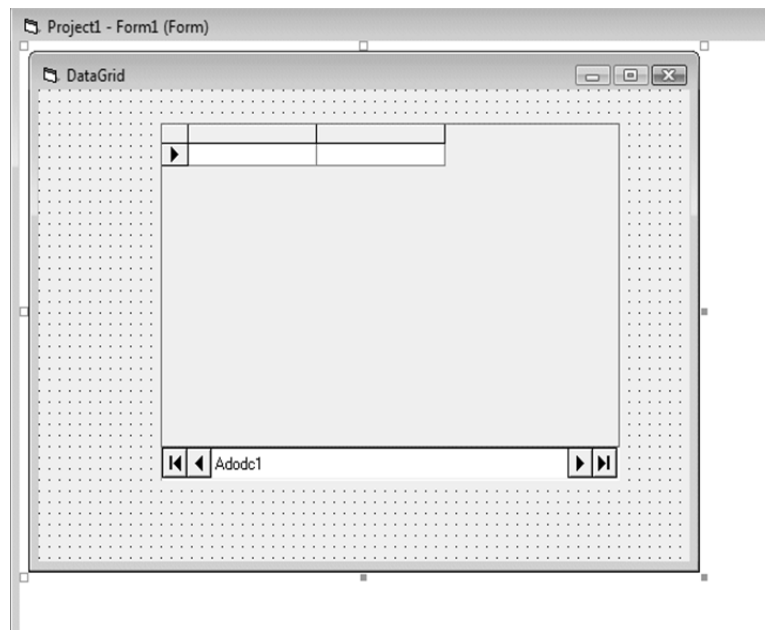
---

The textbox is not the only control that can display data from a database, many other controls in VB can also display data. One of them is the *DataGrid control*. DataGrid control can be used to display the entire table of a recordset of a database. It allows users to view and edit data. DataGrid control is not the default item in the Visual Basic control toolbox, you have to add it from the VB6 components. To add the DataGrid control, click on the Project on the menu bar and select components to access the dialog box that displays all the available VB6

components, as shown in the Fig 5.18. Select Microsoft DataGrid Control 6.0 by clicking the checkbox beside this item. Before you exit the dialog box, you also need to select the Microsoft ADO data control so that you are able to access the database. Last, click on the OK button to exit the dialog box. Now you would be able to see that the DataGrid control and the ADO data control are added to the toolbox. The next step is to drag the DataGrid control and the ADO data control into the form.



**Fig. 5.18 The Component Dialog Box**



**Fig. 5.19 The Design Interface**



## CHECK YOUR PROGRESS

- Describe the picture box control.
- Write the syntax for command button and text box control briefly.
- Give the use of DataGrid control.

---

## 5.6 RICH TEXT BOX CONTROL

---

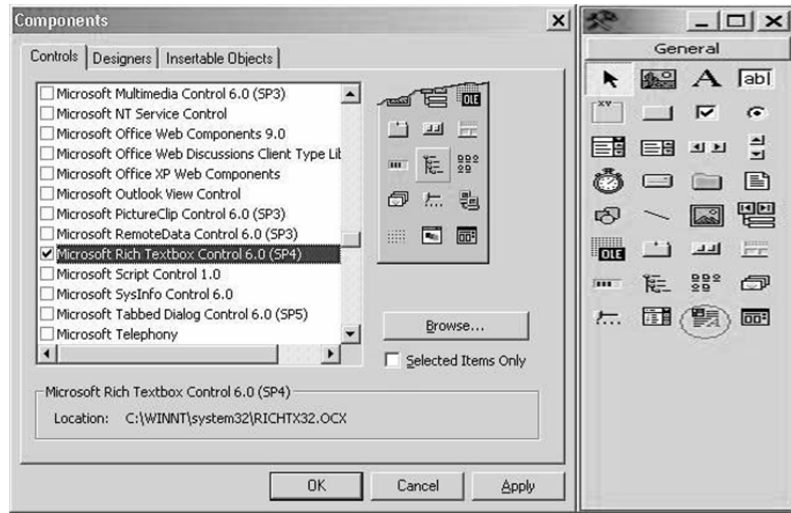
The *RichTextBox* control allows the user to display, enter, and edit text while also providing more advanced formatting features than the conventional text box control. The *RichTextBox* control provides a number of properties you can use to apply formatting to any portion of text within the control. To change the formatting of text, it must first be selected. Only selected text can be assigned character and paragraph formatting. Using these properties, you can make text bold or italic, change the color, and create superscripts and subscripts. You can also adjust paragraph formatting by setting both left and right indents, as well as hanging indents. The *RichTextBox* control opens and saves files in both the RTF format and regular ASCII text format. You can use methods of the control (*LoadFile* and *SaveFile*) to directly read and write files, or use properties of the control such as *SelRTF* and *TextRTF* in conjunction with Visual Basic's file input/output statements.

The *RichTextBox* control supports object embedding by using *OLEObjects* collection. Each object inserted into the control is represented by an *OLEObject* object. This allows you to create documents with the control that contain other documents or objects. For example, you can create a document that has an embedded Microsoft Excel spreadsheet or a Microsoft Word document or any other OLE object registered on your system. To insert objects into the *RichTextBox* control, you simply drag a file (from the Windows Explorer for example), or a highlighted portion of a file used in another application (such as Microsoft Word), and drop the contents directly onto the control.

The *RichTextBox* control supports both clipboard and OLE drag/drop of OLE objects. When an object is pasted in from the clipboard, it is inserted at the current insertion point. When an object is dragged and dropped into the control, the insertion point will track the mouse cursor until the mouse button is released, causing the object to be inserted. This behavior is the same as MS Word. To print all or part of the text in a *RichTextBox* control *SelPrint* method is used.

As *RichTextBox* is a data-bound control, you can bind it with a Data control to a Binary or Memo field in a Microsoft Access database or a similar large capacity field in other databases (such as a TEXT data type field in SQL Server). It supports almost all of the properties, events and methods used with the standard *TextBox* control, such as *MaxLength*, *MultiLine*, *ScrollBars*, *SelLength*, *SelStart*, and *SelText*. Applications that already use *TextBox* controls can easily be adapted to make use of *RichTextBox* controls. However, the *RichTextBox* control doesn't have the

same 64K character capacity limit of the conventional TextBox control. The RichTextBox control must be added to the toolbox via the *Components dialog box*, as shown in Fig 5.20 below. This dialog box is accessed via the Project menu, Components item. Once you check "Microsoft Rich Textbox Control 6.0" and click OK, the control is added to your toolbox (also shown below, circled).



**Fig. 5.20 The Components Dialog Box (left) and RichTextBox Control Added to Toolbox (right)**

---

## 5.6.1 PROPERTIES OF THE RICHTEXTBOX CONTROL

---

**Table 5.2 Properties of the RichTextBox**

Property	Description
<b>AutoVerbMenu</b>	Returns or sets a Boolean value that determines if a pop-up menu containing the RTB's verbs (Cut, Copy Paste) is displayed when the user clicks the RTB control with the right mouse button. When this property is set to True, Click events and MouseDown events don't occur when the RTB control is clicked with the right mouse button. In order to display your own menus, the AutoVerbMenu property must be set to False.
<b>BulletIndent</b>	Returns or sets the amount of indent (Integer) used in the RTB when SelBullet is set to True. The units for the value is based on the ScaleMode setting (twips, pixels, etc.) for the form that contains the RTB. The BulletIndent property returns Null if the selection spans multiple paragraphs with different margin settings.

<b>DisableNoScroll</b>	Returns or sets a Boolean value that determines whether scroll bars in the RichTextBox control are disabled. The DisableNoScroll property is ignored when the ScrollBars property is set to 0 (None). However, when ScrollBars is set to 1 (Horizontal), 2 (Vertical), or 3 (Both), individual scroll bars are disabled when there are too few lines of text to scroll vertically or too few characters of text to scroll horizontally in the RichTextBox control.
<b>FileName</b>	Returns or sets the filename of the file loaded into the RichTextBox control. Only the names of text files or valid .rtf files can be specified for this property.
<b>HideSelection</b>	Returns a Boolean value that determines whether selected text appears highlighted when the RTB loses focus.
<b>Locked</b>	Returns or sets a Boolean value indicating whether the contents of the RTB can be edited. (Setting Locked to True makes the contents of the RTB read-only - however, the content can still be modified by code.)
<b>MaxLength</b>	Returns or sets a value (Long) indicating whether there is a maximum number of characters a RichTextBox control can hold and, if so, specifies the maximum number of characters. The default for the MaxLength property is 0, indicating no maximum other than that created by memory constraints on the user's system. Any number greater than 0 indicates the maximum number of characters. Use the MaxLength property to limit the number of characters a user can enter in a RichTextBox. If text that exceeds the MaxLength property setting is assigned to a RichTextBox from code, no error occurs; however, only the maximum number of characters is assigned to the Text property, and extra characters are truncated. Changing this property doesn't affect the current contents of a RichTextBox, but will affect any subsequent changes to the contents.
<b>MultiLine</b>	Returns or sets a Boolean value indicating whether the RTB can accept and display multiple lines of text. (If False, carriage returns are ignored and text is restricted to a single line.) A multiple-line RichTextBox control wraps text as the user types text extending beyond the text box. Scroll bars can be added to a larger RichTextBox control using the ScrollBars property. If no horizontal scroll bar is specified, the text in a multiple-line RichTextBox automatically wraps.

	<p>Note: On a form with no default button, pressing ENTER in a multiple-line RichTextBox control moves the focus to the next line. If a default button exists, you must press CTRL+ENTER to move to the next line.</p> <p>This property is read-only at run-time.</p>
<b>OLEObjects</b>	<p>Every embedded OLE object created in the RichTextBox control is represented in the OLEObjects collection. You can manually add objects to the OLEObjects collection at run time by using the Add method, or by dragging an object from the Windows Explorer into the RichTextBox control.</p> <p>The OLEObjects collection is a standard collection and supports the Add, Item, and Remove methods, as well as the Count property.</p>
<b>RightMargin</b>	<p>The RightMargin property is a Long value used to set the right most limit for text wrapping, centering, and indentation. Centering a paragraph is based on the left most part of the text portion (doesn't include borders) of the RichTextBox control and the RightMargin property. Also, when setting the SelRightIndent property, it will be based on the current setting of the RightMargin property. The default for the RightMargin property is 0 and will cause the control to set text wrapping equal to the right most part of the RichTextBox control so all text is viewable. Note: Setting this property to a very large value will essentially remove WordWrap.</p>
<b>ScrollBars</b>	<p>Returns or sets a value indicating whether the RTB has horizontal or vertical scroll bars. Read-only at run-time. Possible values are 0-rtfNone (default, no scroll bars shown), 1-rtfHorizontal (horizontal scroll bar only), 2-rtfVertical (vertical scroll bar only), or 3-rtfBoth (both horizontal and vertical scroll bars shown).</p> <p>For a RichTextBox control with setting 1 (Horizontal), 2 (Vertical), or 3 (Both), you must set the MultiLine property to True.</p> <p>At run time, the Windows automatically implements a standard keyboard interface to allow navigation in RichTextBox controls with the arrow keys (UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW), the HOME and END keys, and so on.</p> <p>Scroll bars are displayed only if the contents of the RichTextBox extend beyond the control's borders. If</p>

	<p>ScrollBars is set to False, the control won't have scroll bars, regardless of its contents.</p> <p>A horizontal scrollbar will appear only when the RightMargin property is set to a value that is larger than the width of the control. (The value can also be equal to, or slightly smaller than the width of the control.)</p>
--	--

---

## 5.6.2 METHODS OF THE RICHTEXTBOX CONTROL

---

Method	Description
<b>LoadFile</b>	<p>Loads an .rtf file or text file into a RichTextBox control.</p> <p>Syntax: <i>RichTextBox1.LoadFile pathname, filetype</i></p> <p>Where <i>pathname</i> is a string defining the path and filename of the file to load into the control and <i>filetype</i> is an integer or constant that specifies the type of file loaded. Valid values for <i>filetype</i> are 0 (or the constant <b>rtfRTF</b>; default, specifies a valid .rtf file), or 1 (constant <b>rtfText</b>, specifies any text file).</p> <p>When loading a file with the <b>LoadFile</b> method, the contents of the loaded file replaces the entire contents of the <b>RichTextBox</b> control. This will cause the values of the <b>Text</b> and <b>RTFText</b> properties to change.</p>
<b>SaveFile</b>	<p>Saves the contents of a RichTextBox control to a file.</p> <p>Syntax: <i>RichTextBox1.SaveFile pathname, filetype</i></p> <p>Where <i>pathname</i> is a string defining the path and filename of the file to receive the contents of the control and <i>filetype</i> is an integer or constant that specifies the type of file to be saved. Valid values for <i>filetype</i> are 0 (or the constant <b>rtfRTF</b>; default, saves contents as a .rtf file), or 1 (constant <b>rtfText</b>, saves contents as a text file).</p>

---

## 5.7 WORKING WITH MENUS

---

Menus, which are located on the menu bar of a form, contain a list of related commands. When you click a menu title in a Windows-based program, a list of menu commands should always appear in a well-organized list. Most menu commands run immediately after they are clicked. For example, when the user clicks the Edit menu's copy command, Windows immediately copies information to the Clipboard. However, if ellipsis points (...) follow the menu command, Visual Basic displays a dialog box that requests more information before the command is carried out.

This section includes the following topics:

- Using the Menu Editor
- Adding Access and Shortcut Keys
- Processing Menu Choices

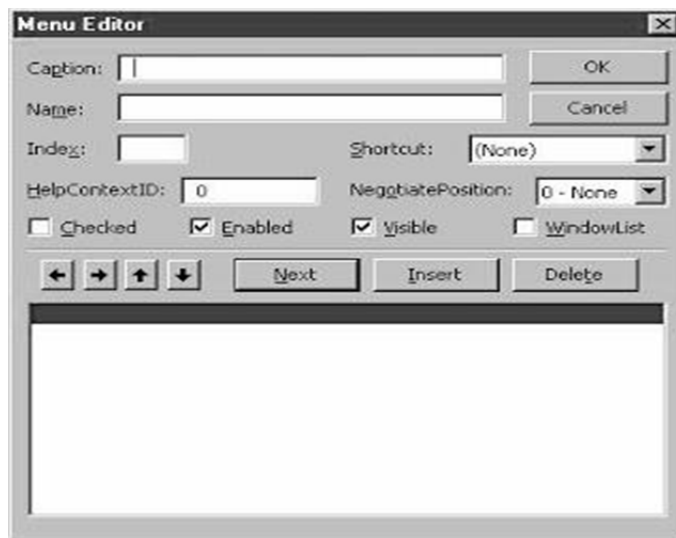
---

## 5.7.1 USING THE MENU EDITOR

---

The Menu Editor is a VB dialog box that manages menus in your programs. With the Menu Editor, you can:

- Add new menus
- Modify and reorder existing menus
- Delete old menus
- Add special effects to your menus, such as access keys, check marks, and keyboard
- Shortcuts.



**Fig. 5.21 The Menu Editor**

---

### 5.7.1.1 CREATING MENU COMMAND LISTS

---

To build lists of menu commands, you first need to create the menus and then add them to the program menu bar.

To create a list of menu commands on a form

1. Click the form itself (not an object on the form).
2. On the Visual Basic toolbar, click the Menu Editor icon, or select Menu Editor from the Tools menu.

3. In the Caption text box, type the menu caption (the name that will appear on the menu bar), and then press TAB.
4. In the Name text box, type the menu name (the name the menu has in the program code). By convention, programmers use the mnu object name prefix to identify both menus and menu commands.
5. To add the menu to your program menu bar, click Next. The Menu Editor clears the dialog box for the next menu item. As you build your menus, the structure of the menus and commands appear at the bottom of the dialog box.
6. In the Caption text box, type the caption of your first menu command.
7. Press tab, and then type the object name of the command in the Name text box.
8. With this first command highlighted in the menu list box, click the right arrow button in the Menu Editor. In the Menu list box, the command moves one indent (four spaces) to the right. Click the right arrow button in the Menu Editor dialog box to move items to the right, and click the left arrow button to move items to the left.
9. Click Next, and then continue to add commands to your menu.

---

### **5.7.1.2 TO ADD MORE MENUS**

---

1. When you're ready to add another menu, click the left arrow button to make the menu flush left in the Menu list box.
2. To add another menu and menu commands, repeat Steps 3 through 9 in the preceding procedure.
3. When you're finished entering menus and commands, click OK to close the Menu Editor. (Don't accidentally click Cancel or all your menu work will be lost.) The Menu Editor closes, and your form appears in the programming environment with the menus you created.

---

### **5.7.1.3 ADDING EVENT PROCEDURES**

---

After you add menus to your form, you can use event procedures to process the menu commands. Clicking a menu command on the form in the programming environment displays the event procedure that runs when the menu command is chosen. You'll learn how to create event procedures that process menu selections in Processing Menu Choices.

---

## **5.7.2 ADDING ACCESS AND SHORTCUT KEYS**

---

Visual Basic makes it easy to provide access key and shortcut key support for menus and menu commands.

---

### 5.7.2.1 ACCESS AND SHORTCUT KEYS

---

The access key for a command is the letter the user can press to execute the command when the menu is open. The shortcut key is the key combination the user can press to run the command without opening the menu. Here's a quick look at how to add access and shortcut keys to existing menu items:

1. Add an access key to a menu item Start the Menu Editor.
2. Prefix the access key letter in the menu item caption with an ampersand (&).
3. Add a shortcut key to a menu command Start the Menu Editor.
4. Highlight the command in the menu list box.
5. Pick a key combination from the Shortcut drop-down list box.

---

### 5.7.2.2 CREATING ACCESS AND SHORTCUT KEYS

---

You can create access keys and shortcut keys either when you first create your menu commands or at a later time. The following illustration shows the menu commands associated with two menus, File and Clock. Each menu item has an access key ampersand character, and the Time and Date commands are assigned shortcut keys (Fig.5.22).

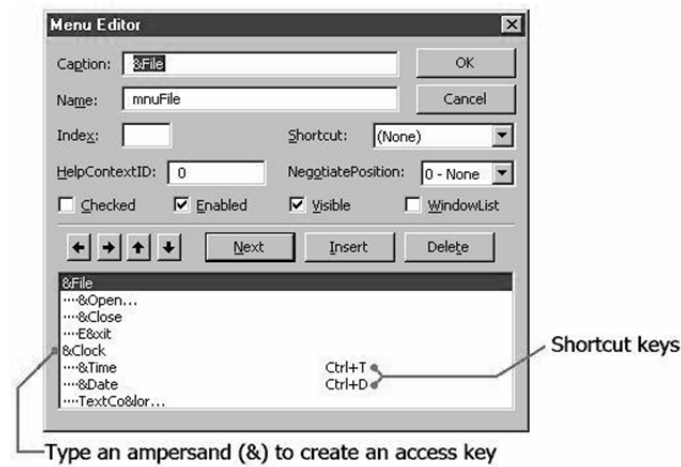


Fig. 5.22 The Menu Editor

---

### 5.7.3 PROCESSING MENU CHOICES

---

In a typical Windows application, not all menu commands are available at the same time. In a typical Edit menu, for example, the Paste command is available only when there is data on the Clipboard. When a command is disabled, it appears in dimmed (gray) type on the menu bar. You can disable a menu item by:

1. Clearing the Enabled check box for that menu item in the Menu Editor.



2. Using program code to set the item's Enable property to False. (When you're ready to use the menu command again, set its Enable property to True.)

---

## 5.8 ADDING AND REMOVING A CONTROL ON FORM

---

You can add controls to a form in two ways: by double-clicking and by drawing. You learned about double-clicking earlier, whenever you double-click an icon on the toolbar, the associated control appears on your form. When you do this, though, you can't control where the control goes: You're at the mercy of VB. When you draw a control on your form, you can put it wherever you want it.

---

### 5.8.1 DRAW A CONTROL ON A FORM

---

1. Click the control's Toolbox icon.
2. Move the mouse pointer over your form. Notice that your pointer is now shaped as a crosshair instead of an arrow. Click (and hold) the mouse button where you want the control to go.
3. Drag the mouse down slightly and to the left. As you move the mouse, notice that a box starts to appear.
4. When the box is the proper size, let go of the mouse button. The control you selected now appears on the form.

---

### 5.8.2 REMOVE A CONTROL FROM A FORM

---

1. Select the control you want to delete by clicking it. The control you select will appear with a box at each corner and side.
2. Press the Delete key.

You can also remove a control by right-clicking it. From the context menu that appears, select Delete.

---

## 5.9 NAMING A CONTROL

---

A control name is one of its most important attributes because you literally refer to a control by its name whenever you want it to do something. Names are so important that every time you put a control on your form, VB automatically gives a name to it. If you add a command button, VB names it Command1; if you add a text box, it's automatically named Text1. However, naming controls like this can be confusing. For example, if you add six command buttons to your form, VB gives them names as Command1, Command2, Command3, and so on. If you need 100 buttons, VB assigns a name Command100 to the last one. How are you supposed to remember what Command67 does? The trick is, you should assign names yourself instead of VB doing it automatically.

### Name a Control:

1. After you add a control to a form, make sure that it's selected (it has a box at each corner and side when it's selected).
2. In the Properties window, click the control's name in the right column.
3. Delete the current name and add the name you want.

A better name for a control is one that tells not only what type of control it is, but also what it does within your program. Can you see the value here? If you consistently give your controls descriptive names, you'll always know what they do. Naturally, there is a convention you can use to help you with this.

When naming a control, the first letter of the friendly name is generally uppercase. This makes it easier to read the control's name, because you can easily differentiate between the friendly name and the control's abbreviation. This convention is quite simple. It consists of a short (usually three-letter) abbreviation that identifies the type of control (Table 5.2), followed by a longer, friendly name that describes what the control does within your program. The abbreviation is lowercase, and the friendly name follows it immediately, without any spaces.

**Table 5.2 Common Prefixes for Naming Visual Basic Intrinsic Controls**

<i>Control</i>	<i>Prefix</i>	<i>Control</i>	<i>Prefix</i>
Label	Lbl	PictureBox	Pic
Frame	Fra	TextBox	Txt
CheckBox	Chk	CommandButton	Cmd
ComboBox	Cbo	OptionButton	Opt
HscrollBar	Hsb	ListBox	Lst
Timer	Tmr	VscrollBar	Vsb
DirListBox	Dir	DriveListBox	Drv
Shape	Shp	FileListBox	Fil
Image	Img	Line	Lin
OLE Container Control	Ole	Data	Dat

## CHECK YOUR PROGRESS

- Compare RichTextBox with standard TextBox control.
- Write the significance of naming a control.
- Write the common prefixes for naming Visual Basic Intrinsic Controls.

---

## 5.10 SUMMARY

---

A *Control* is an object that enables users to interact with the application. Controls are available in tool box. These are the building blocks of an application. Each control has a unique set of properties, which determine its appearance and behavior. A control has a name property that distinguishes it from other objects in a project. There are other properties as well which can be changed.

The *Custom control* is a program that someone has written which can be included in the VB program. The two types of custom controls that VB can use are VBX (Visual Basic custom extension controls) and OCX (OLE Custom extension controls). Moreover, a custom control is a file with a .VBX or .OCX filename extension or an insertable object that can be added to a project using the custom controls dialog box to extend the 'Toolbox'. The desired selection can be made by clicking the box to the left of the list, which shall mark 'X' in the box. The deselection shall remove the 'X' mark.

The TextBox is the standard control for accepting input from the user as well as to display the output. It can handle string (text) and numeric data but not images or pictures. A string entered into a text box can be converted to a numeric data by using the function Val(text). The CommandButton is one of the most important controls as it is used to execute commands. It displays an illusion that the button is pressed when the user clicks on it. The most common event associated with the command button is the click event. The procedure associated with the command button is *Comman1\_click()*.

The PictureBox is one of the controls that is used to handle graphics. A picture can be loaded at design phase by clicking on the picture item in the properties window and select the picture from the selected folder. The picture can be loaded at runtime using the *LoadPicture* method. The ImageControl is another control that handles images and pictures. It functions almost identically to the PictureBox control. However, there is one major difference, the image in an ImageControl is stretchable, which means it can be resized. This feature is not available in the PictureBox. Similar to the PictureBox, it can also use the LoadPicture method to load the picture.

The function of the ListBox is to present a list of items where the user can click and select the items from the list. In order to add items to the list, we can use the AddItem method. The function of the ComboBox is also to present a list of items where the user can click and select the items from the list. However, the user needs to click on the small arrowhead on the right of the combo box to see

the items which are presented in a drop-down list. In order to add items to the list, *AddItem* method can also be used. The *CheckBox* control lets the user select or unselect an option. When the check box is checked, its value is set to 1 and when it is unchecked, the value is set to 0. You can include the statements *Check1.Value = 1* to mark the Check Box and *Check1.Value = 0* to unmark the Check Box, as well as use them to initiate certain actions.

The *OptionButton* control also lets the user select one of the choices. However, two or more option buttons must work together because as one of the option buttons is selected, the other Option button will be unselected. In fact, only one Option Box can be selected at one time. When an option box is selected, its value is set to "True" and when it is unselected; its value is set to "False". The *Shape* control is used to determine the shape of the shape using the shape property. The property values of the shape control are 0, 1, 2, 3, 4, 5 which will make it appear as a rectangle, a square, an oval, a circle, a rounded rectangle, and a rounded square respectively.

The *DriveListBox* is for displaying a list of drives available in the computer. When you place this control into the form and run the program, you will be able to select different drives from your computer. The *DirListBox* means the Directory List Box. It is for displaying a list of directories or folders in a selected drive. When you place this control into the form and run the program, you will be able to select different directories from a selected drive in your computer.

A *Timer* control is invisible at run time, and its purpose is to send a periodic pulse to the current application. You can trap this pulse by writing code in the Timer's timer event procedure and take advantage of it to execute a task in the background or to monitor a user's actions. This control exposes only two meaningful properties: Interval and Enabled.

The *TextBox* is not the only control that can display data from a database, many other controls in VB can display data. One of them is the *DataGrid control*. DataGrid control can be used to display the entire table of a recordset of a database. The *RichTextBox* control allows the user to display, enter, and edit text while also providing more advanced formatting features than the conventional *TextBox* control. The RichTextBox control provides a number of properties can be used use to apply formatting to any portion of text within the control. To change the formatting of text, it must first be selected. Only selected text can be assigned character and paragraph formatting.

Controls to a form can be added in two ways: First, by double-clicking, and second by drawing. Whenever you double-click an icon on the toolbar, the associated control appears on the form. When you do this, though, you can't control where the control goes: You're at the mercy of VB. When you draw a control on your form, you can put it wherever you want it. A control's name is one of its most important attributes because you literally refer to a control by its name whenever you want it to do something. Names are so important that every time you put a control on your form, VB automatically gives a name to it. If you add a command button, VB names it *Command1*; if you add a *TextBox*, it automatically names *Text1* and so on.

---

## 5.11 TERMINAL QUESTIONS

---

1. What do you understand by control and its related properties? Explain briefly.
2. Compare Intrinsic control and Custom control.
3. What do you understand by Custom control? How can these be added on a form?
4. Write a short note on Intrinsic controls.
5. Differentiate standard TextBox and RichTextBox.
6. What is the difference between PictureBox control and ImageControl?
7. What do you understand by a RichTextBox control? Explain its properties.
8. Write a short note on the addition and removal of a control.
9. Discuss the importance of naming a control.
10. Write a short note on Text and Command control.



---

# UNIT-6 DIALOG BOXES AND INTERNET

---

## Structure

- 6.0 Introduction
- 6.1 Objectives
- 6.2 Introduction to Dialog Boxes
- 6.3 Modal Dialog Box
- 6.4 Modeless Dialog Box
- 6.5 Model Vs Modeless Dialog Box
- 6.6 Common Dialog Box
- 6.7 Visual Basic and Internet
- 6.8 Summary
- 6.9 Terminal Questions

---

## 6.0 INTRODUCTION

---

This Unit 6 basically deals with the dialog boxes used in VB. As there are so many dialog boxes used in various applications like MS office, Notepad, Wordpad etc. Some popular examples of the dialog boxes are like file open dialog box, font dialog box, save as dialog box, print dialog box, message dialog box. Usually, the dialog boxes are divided in two categories *modal* and *modeless* dialog boxes. Control box basically includes various controls like text boxes, command buttons, check boxes etc.

A dialog box is simply a form in a program that contains input controls designed to receive information. To make your programming faster, Visual Basic includes an ActiveX control, named CommonDialog. With this control, you can easily display six standard dialog boxes in your programs.

Two important categories of dialog boxes are modal and modeless. A modal dialog box is basically a window that forces the user to interact with it before they can go back to using the parent application. A good example of this would be a prompt for saving, or the "open file" dialog. Modal dialog boxes, which require the user to respond before continuing the program or forces the user to acknowledge the dialog before moving onto the application. On the other hand, modeless dialog boxes enable the user to interact with the dialog box and the application interchangeably.

Moreover, some controls are not available in the control box or tool box, but they can be added as per the need. Web browser is one of the controls which can be added as per the need of the application. Like an Internet based application require the web browser control. Furthermore, apart from the traditional dialog

boxes, the *Common Dialog Control* provides a standard interface for operations such as opening, saving, and printing files or selecting colours and fonts using the Microsoft Windows dynamic link library COMMDLG.

---

## 6.1 OBJECTIVES

---

At the end of this unit you will come to know about the following

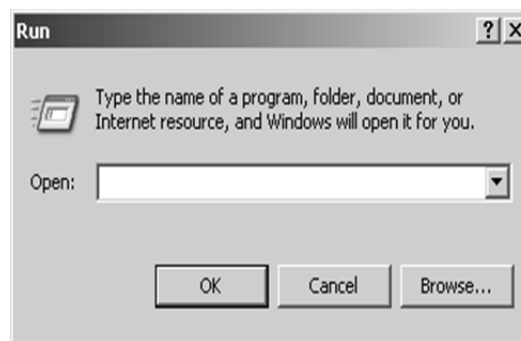
- Dialog box
- Message dialog box
- Modal dialog box
- Modeless dialog box
- Common dialog box
- Font dialog box
- Print dialog box
- Save as dialog box
- File open dialog box
- Relation between Internet and VB

---

## 6.2 INTRODUCTION TO DIALOG BOXES

---


A dialog box is a form defined with particular properties. Like a form, a dialog box is referred to as a container. A dialog box is mostly used to host child controls, insuring the role of dialog between the user and the machine. A dialog box is simply a form in a program that contains input controls designed to receive information. To make your programming faster, VB includes an ActiveX control, named Common Dialog control. With this control, you can easily display six standard dialog boxes in the programs. These dialog boxes handle routine tasks such as opening files, saving files, picking fonts, printing etc. If the dialog box you want to use is not included in this ready-made collection of objects, you can create a new one by adding a second form to your program. Here is an example of a dialog box in Fig. 6.1.



**Fig. 6.1 The Dialog Box**

A dialog box has the following characteristics:



- The only system button is equipped with is Close . As the only system button, this button allows the user to dismiss the dialog and ignore whatever the user would have done on the dialog box.
- It cannot be minimized, maximized, or restored. A dialog box does not have any other system button but Close.
- It is usually modal, in which case the user is not allowed to continue any other operation on the same application until the dialog box is dismissed.
- It provides a way for the user to close or dismiss it.

## 6.2.1 CREATING A DIALOG BOX USING THE COMMON DIALOG CONTROL

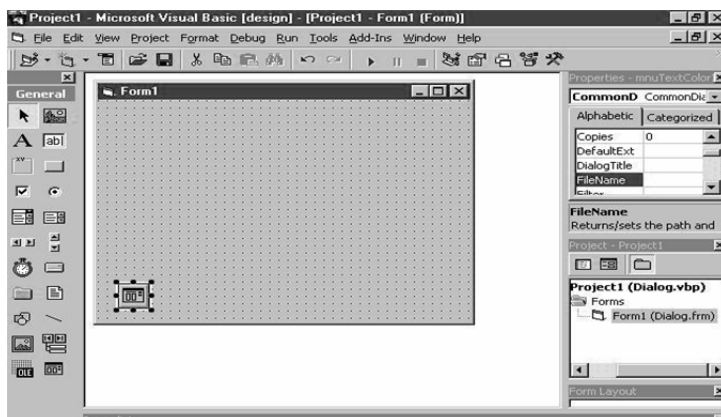
Before you can use the Common Dialog control, you need to verify that it is in your toolbox. If you don't see the Common Dialog icon, follow the following procedure to add it to the toolbox. To add the Common Dialog control to the toolbox, use the following steps:

1. From the Project menu, click Components.
2. Click the Controls tab.
3. Ensure that the Selected Items Only box is not checked.
4. Place a check mark next to Microsoft Common Dialog Control, and then click OK.

Follow this procedure to create a dialog box with the CommonDialog control:

1. In the toolbox, double-click the Common Dialog control.
2. When the common dialog object appears on your form (Fig. 6.2), drag it to an out-of-the-way location.

**Note:-** You cannot resize a common dialog object, and it disappears when your program runs. The common dialog object itself displays nothing — its only purpose is to display a standard dialog box on the screen when you use a method in program code to request it.



**Fig. 6.2 Adding Common Dialog Control**

This following table lists the name and purpose of the six standard dialog boxes that the common dialog object provides and the methods you use to display them.

**Table 6.1 List of standard dialog boxes**

<b>Dialog Box</b>	<b>Purpose</b>	<b>Method</b>
Open	Gets the drive, folder, name, and file name for an existing file that is being opened.	ShowOpen
Save As	Gets the drive, folder name, and file name for an existing file that is saved.	ShowSave
Print	Provide user-defined printing options.	ShowPrinter
Font	Provide user-defined font type and style options.	ShowFont
Help	Provide online user information.	ShowHelp
Color	Provide user-defined color selection from a palette.	ShowPrinter

---

## **6.2.2 USING THE DIALOG OBJECT EVENT PROCEDURES**

---

To display a standard dialog box in a program, you need to call the common dialog object. You do this by using the appropriate object method in an event procedure. If necessary, you also use program code to set one or more common dialog object properties before the call. (For example, if you are using the Open dialog box, you might want to control what type of files is displayed in the list box.) Finally, your event procedure needs to process the choices made by the user when they complete the standard dialog box. This section presents two simple event procedures, one that manages an Open dialog box and one that uses information received from a Color dialog box.

---

### **6.2.2.1 CREATING AN OPEN DIALOG BOX**

---

The following code window shows an event procedure named *mnuOpenItem\_Click*. You can use this event procedure to display an Open dialog box when the user clicks the Open command on the File menu. The event procedure assumes that you have already created a File menu containing Open and Close commands and that you want to open Windows metafiles (.wmf). See the piece of code given below.

```
Private Sub mnuOpenItem_Click()
    CommonDialog1.Filter = "Metafiles (*.WMF)|*.WMF"
```

```

CommonDialog1.ShowOpen
Image1.Picture = LoadPicture(CommonDialog1.FileName)
mnuCloseItem.Enabled = True

End Sub

```

The event procedure uses the following properties and methods shown in table 6.2.

**Table 6.2 List of Property and Methods by Event Procedures**

Object	Property/Method	Purpose
Common Dialog	ShowOpen	Displays the dialog box.
Common Dialog	Filter	Defines the file type in the dialog box.
Menu	Enabled	Enable the Close menu command, which users can use to unload the picture.
Image	Picture	Opens the selected file.

---

### 6.2.2.2 CREATING A COLOR DIALOG BOX

---

If you need to update the colour of a user interface element while your program runs, you can prompt the user to pick a new colour with the Color dialog box displayed by using the Common Dialog object. The color selections provided by the Color dialog box are controlled by the Flags property, and the Color dialog box is displayed with the ShowColor method.

This code window shows an event procedure that you can use to change the color of a label while your program runs. The value used for the Flags property—which in this case prompts VB to display a standard palette of color selections—is a hexadecimal (base 16) number. (To see a list of other potential values for the Flags property, search for CommonDialog constants in the VB online Help.) The event procedure assumes that you have already created a menu command named TextColor with the Menu Editor. See the code given below

```

Private Sub mnuTextColorItem_Click()

CommonDialog1.Flags = &H1&

CommonDialog1.ShowColor
Label1.ForeColor = CommonDialog1.Color

End Sub

```

**Note:** - Normally there are two types of dialog boxes- modal and modeless dialog box which are discussed in detail below.

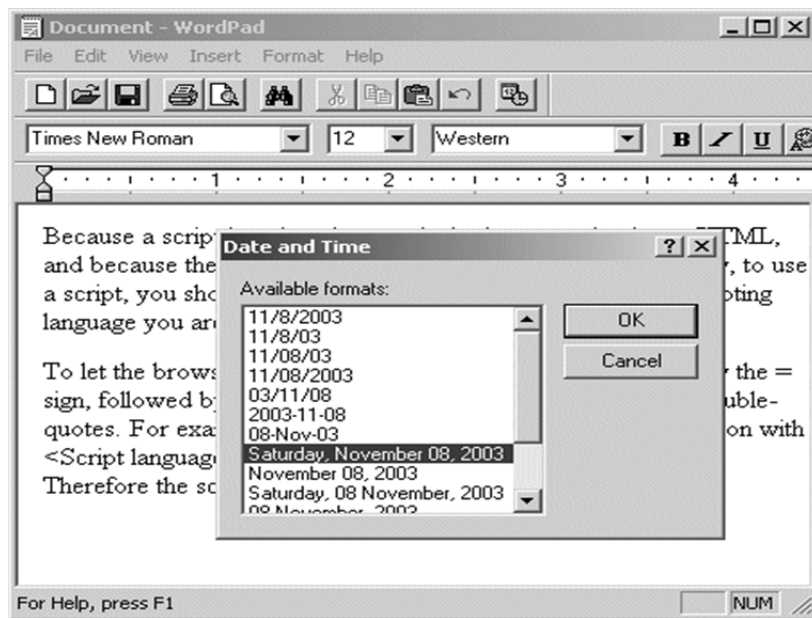
---

## 6.3 MODAL DIALOG BOX

---

A modal dialog box is a window that forces the user to interact with it before they can go back to using the parent application. Modal dialog boxes force the user to acknowledge the dialog before moving on to the application. They are often used when a user is forced to make an important decision. For example, if you are working on a document in Microsoft Word, and you want to choose to exit Word before saving. A modal dialog would pop up and ask you to either Save, Don't Save, or Cancel. Until you make your decision you cannot use the application, and it will not close. A Modal dialog box is one that the user must first close in order to have access to any other framed window or dialog box of the same application. One of the scenarios in which you use a dialog box is to create an application that is centered around one. In this case, if either there is no other form or dialog box in your application or all the other forms or dialog boxes depend on this central dialog box, it must be created as modal. Such an application is referred to as dialog-based.

Modal dialogs should be used when the user takes an action that needs additional information to be completed correctly. The print dialog box is also a good example of modal dialog box or the 'options' dialogs in loading of programs. When to use modal dialogs is a bit tricky to define, but mostly it should be used sparingly, and avoided whenever possible. Some applications require various dialog boxes to complete their functionality. When in case, you may need to call one dialog box from another and display it as modal. Here is an example in Fig. 6.3.



**Fig. 6.3 The Date and Time dialog box of WordPad is modal: when it is displaying, the user cannot use any other part of WordPad unless he or she closes this object first**

---

### 6.3.1 EXAMPLES

---

1. A SSH program requires the password to connect to the server. That *may* be implemented via a modal dialog, though you could argue that opening the options configuration makes sense whether the user is connected or not.
2. A multi-tabbed browser should *not* implement HTTP 401 basic authentication via a modal dialog. A user may well want to open another tab to read the password from an activation email.
3. A print dialog in a multi-document interface should not be modal. If the user wants to print two documents, one with 1000 pages and one with 5, but wrongly starts with the 1000-page document, you should not require him/her to cancel the dialog (and all settings he/she gave) so that the smaller job could be printed first.

---

## 6.4 MODELESS DIALOG BOX

---

*Modeless* dialog boxes enable the user to interact with the dialog and the application interchangeably. So far we've been looking at modal dialog boxes, the more common of the two types. Modal dialog boxes (except system modal dialog boxes) allow the user to switch between the dialog box and other programs. However, the user cannot switch to another window in the program that initiated the dialog box until the modal dialog box is destroyed. Modeless dialog boxes allow the user to switch between the dialog box and the window that created it as well as between the dialog box and other programs. The modeless dialog box is thus more akin to the regular popup windows that your program might create.

Modeless dialog boxes are preferred when the user would find it convenient to keep the dialog box displayed for a while. For instance, word processors often use modeless dialog boxes for the text Find and Change dialogs. If the Find dialog box were modal, the user would have to choose Find from the menu, enter the string to be found, end the dialog box to return to the document, and then repeat the entire process to search for another occurrence of the same string. Allowing the user to switch between the document and the dialog box is much more convenient.

As it has been seen, modal dialog boxes are created using *DialogBox*. The function returns a value only after the dialog box is destroyed. It returns the value specified in the second parameter of the *EndDialog* call that was used within the dialog box procedure to terminate the dialog box. Modeless dialog boxes are created using *CreateDialog*. This function takes the same parameters as *DialogBox*:

```
hDlgModeless = CreateDialog (hInstance, szTemplate,  
                             hwndParent, DialogProc);
```

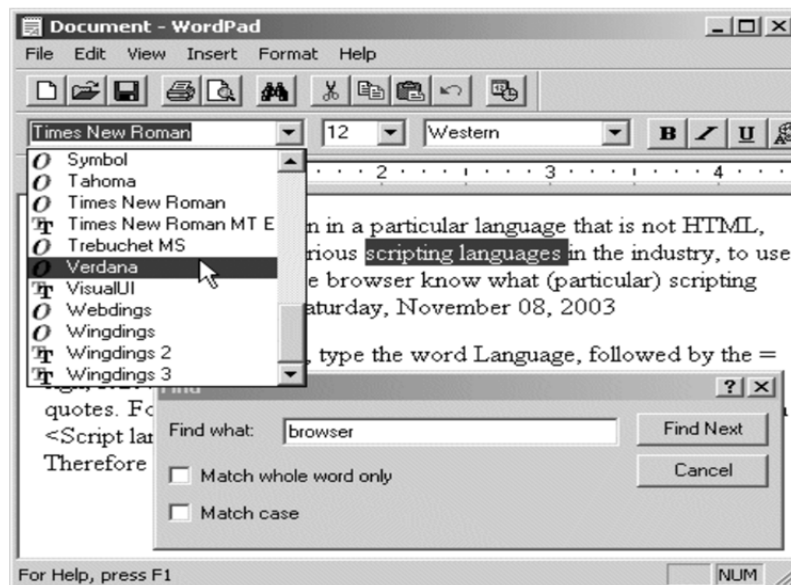
The difference is that the *CreateDialog* function returns immediately with the window handle of the dialog box. Normally, you store this window handle in a

global variable. Although the use of the names *DialogBox* with modal dialog boxes and *CreateDialog* with modeless dialog boxes may seem arbitrary, you can remember which is which by keeping in mind that modeless dialog boxes are similar to normal windows. *CreateDialog* should remind you of the *CreateWindow* function, which creates normal windows.

A dialog box is referred to as modeless if the user does not have to close it in order to continue using the application that owns the dialog box. A modeless dialog box has the following characteristics:

- It has a thin border
- It can be neither minimized nor maximized. This means that it is not equipped with the Minimize or the Maximize buttons
- It is not represented on the taskbar with a button
- It must provide a way for the user to close it

**Example:** The Find (and the Replace) dialog box of WordPad (also the Find and the Replace dialog boxes of most applications) is an example of a modeless dialog box. If it is opened, the user does not have to close it in order to use the application or the document in the background (Fig. 6.4).



**Fig. 6.4 The Find (and the Replace) dialog box of WordPad**

Since the modeless dialog box does not display its button on the task bar, the user should know that the dialog box is opened. To make the presence of a modeless dialog box obvious to the user, it typically displays on top of its host application until the user closes it.

A modeless dialog box is created from a form but it should look like a regular dialog box or a tool window. Therefore, to create a modeless dialog box, set the *FormBorderStyle* property to an appropriate value such

as `FixedSingle`, `FixedToolWindow`, `Sizable` or `SizableToolWindow`. Also, set its `ShowInTaskbar` property to `False`. After creating the dialog box, to display it as modeless, call the `Show()` method. The fundamental difference between the `ShowDialog()` and the `Show()` methods is that the former displays a modal dialog box, which makes sure that the called dialog box cannot go in the background of the main application. By contrast, the `Show()` method only calls the dialog box every time it is requested. For this reason, it is up to you to make sure that the modeless dialog box always remains on top of the application. This is easily taken care of by setting the boolean `TopMost` property of the form to `True`.

There are two main ways a normal modeless dialog box can be dismissed:

- If the user has finished using it, he or she can close it and recall it at will
- When the form or application that owns the modeless dialog box is closed, the form or application closes the modeless dialog if it is opened; this means that you don't need to find out whether a modeless dialog box is still opened when the application is being destroyed: either the user or the application itself will take care of closing it.

---

## 6.5 MODAL VS MODELESS DIALOG BOX

---

- *Modal* dialog boxes force the user to acknowledge the dialog before moving onto the application. *Modeless* dialog boxes enable the user to interact with the dialog and the application interchangeably.
- A modal dialog box doesn't allow the user to access the parent window while the dialog is open – it must be dealt with and closed before continuing. A modeless dialog can be opened in the background.
- Modal dialog boxes, which require the user to respond before continuing the program.
- Modeless dialog boxes, which stay on the screen and are available for use at any time but permit other user activities.
- When a modal dialog is open you cannot interact with anything else than this modal dialog inside your program, as long as the modal dialog is open. Most dialogs are modal, for example the File-Save As dialogs are modal.
- On the other hand, a modeless dialog behaves just like a normal window, you can do anything you want while it is open. The spell checker dialog in Microsoft Word is an example of such a dialog
- Modal dialog box captures the message loop. Whereas modeless does not.
- Modal dialog does not allow its parent window unless it is closed. Whereas modeless it allows.
- Modal Dialog box occupies the Stack Area that the reason it can't give the control to its parent. Modeless Dialog box occupies the Heap Area it gives the control to its parent.
- Modal dialog does not switch the control of dialog box outside the window.

- Modeless dialog can perform actions outside dialog box of the window.
- Model dialog box is a static for the model box control application, & the modeless dialog box is a dynamic, so you can do anything in modeless dialog box.

**CHECK YOUR PROGRESS**

- What do you understand by dialog box?
- Compare modal and modeless dialog box.
- Write at least one example for modal and modeless dialog box each.

---

## 6.6 COMMON DIALOG BOXES

---

The Common Dialog Box Library contains a set of dialog boxes for performing common application tasks, such as opening files, choosing color values, and printing documents. The common dialog boxes allow you to implement a consistent approach to your application's user interface. This reduces the amount of effort that users spend in learning user interface behavior for your application.

To give the user a standard interface for common operations in Windows-based applications, Visual Basic provides a set of common dialog boxes, two of which are the Open and Save As dialog boxes. Such boxes are familiar to any Windows user and give your application a professional look. And, with Windows, some context-sensitive help is available while the box is displayed.

The Common Dialog control is a 'custom control' which means we have to make sure some other files are present to use it. In normal setup configurations, Visual Basic does this automatically. If the common dialog box does not appear in the Visual Basic toolbox, you need to add it. This is done by selecting Components under the Project menu. When the selection box appears, click on Microsoft Common Dialog Control, then click OK. The common dialog tool, although it appears on your form, is invisible at run-time. You cannot control where the common dialog box appears on your screen. The tool is invoked at run-time using one of five 'Show' methods (Table 6.3).

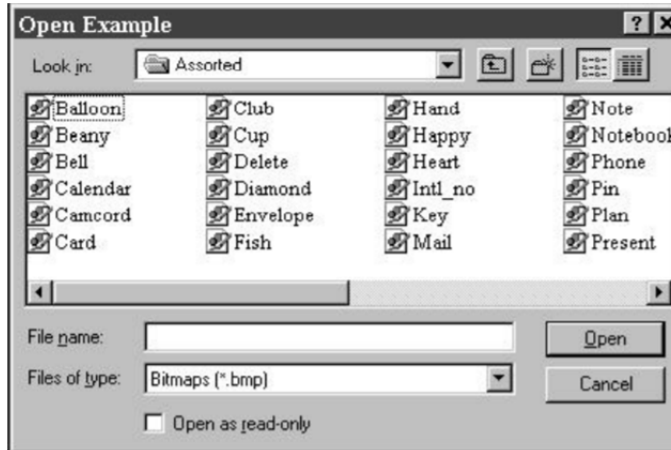
**Table 6.3 Show Methods**

S.No.	Method	Common Dialog Box
1	ShowOpen	Open dialog box
2	ShowSave	Save As dialog box
3	ShowColor	Color dialog box
4	ShowFont	Font dialog box
5	ShowPrinter	Printer dialog box



## 6.6.1 OPENFILEDIALOG BOX

The Open common dialog box provides the user a mechanism for specifying the name of a file to open. The box is displayed by using the ShowOpen method. Here's an example of an Open common dialog box (Fig. 6.5).



**Fig. 6.5 Open Common Dialog Box**

### 6.6.1.1 OPENFILEDIALOG BOX PROPERTIES

OpenFileDialog Box Properties are shown in Table 6.4 and there are six properties.

**Table 6.4 OpenFileDialog Box Properties**

S.No.	Property	Description
1	CancelError	If True, generates an error if the Cancel button is clicked. Allows you to use error-handling procedures to recognize that Cancel was clicked.
2	DialogTitle	The string appearing in the title bar of the dialog box. Default is Open. In the example, the DialogTitle is Open Example.
3	FileName	Sets the initial file name that appears in the File name box. After the dialog box is closed, this property can be read to determine the name of the selected file.
4	Filter	Used to restrict the filenames that appear in the file list box. Complete filter specifications for forming a Filter can be found using on-line help. In the example, the Filter was set to allow Bitmap (*.bmp), Icon (*.ico), Metafile (*.wmf), GIF (*.gif), and JPEG (*.jpg) types (only the Bitmap choice is seen).

5	FilterIndex	Indicates which filter component is default. The example uses a 1 for the FilterIndex (the default value).
6	Flags	Values that control special features of the Open dialog box. The example uses no Flags value.

## 6.6.2 SAVE AS DIALOG BOX

The Save As common dialog box provides the user a mechanism for specifying the name of a file to save. The box is displayed by using the ShowSave method as shown in the Fig. 6.6.

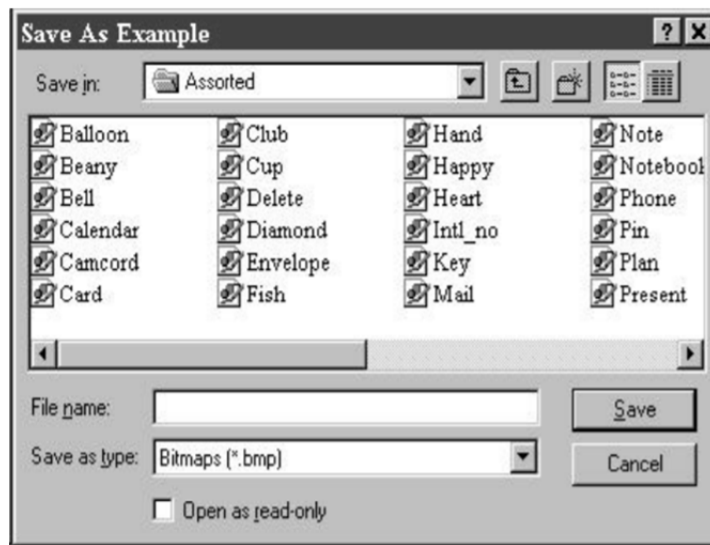


Fig. 6.6 Save As Common Dialog Box

### 6.6.2.1 SAVE AS DIALOG BOX PROPERTIES

Save As Dialog Box Properties are shown in Table 6.5 and there are seven properties.

Table 6.5 Save As Dialog Box Properties

S.No.	Property	Description
1	CancelError	If True, generates an error if the Cancel button is clicked. Allows you to use error-handling procedures to recognize that Cancel was clicked.
2	DefaultText	Sets the default extension of a file name if a file is listed without an extension.

3	DialogTitle	The string appearing in the title bar of the dialog box. Default is Save As. In the example, the DialogTitle is Save As Example.
4	FileName	Sets the initial file name that appears in the File name box. After the dialog box is closed, this property can be read to determine the name of the selected file.
5	Filter	Used to restrict the filenames that appear in the file list box.
6	FilterIndex	Indicates which filter component is default.
7	Flags	Values that control special features of the dialog box

---

### 6.6.2.2 SAVE AS DIALOG BOX METHODS

---

Save As Dialog Box methods are shown in Table 6.6 and there are two properties.

**Table 6.6 Save As Dialog Box Properties**

S.N.	Method Name	Description
1.	OpenFile	Opens the file with read/write permission.
2.	Reset	Resets all dialog box options to their default values.

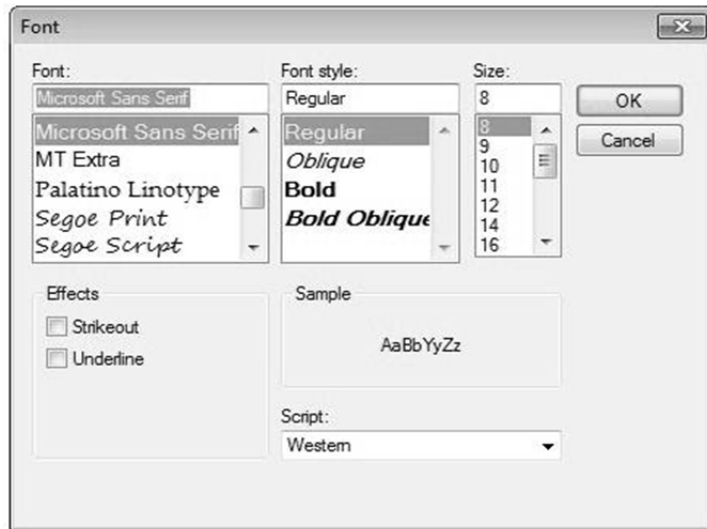
The Save File box is commonly configured in one of two ways. If a file is being saved for the first time, the Save As configuration, with some default name in the FileName property, is used. In the Save configuration, we assume a file has been previously opened with some name. Hence, when saving the file again, that same name should appear in the FileName property. You've seen both configuration types before.

---

### 6.6.3 FONT DIALOG BOX

---

The Font dialog box lets the user choose attributes for a logical font, such as font family and associated font style, point size, effects, and a script. It prompts the user to choose a font from among those installed on the local computer and lets the user select the font size and color. It returns the font and color objects. The font dialog box is shown in Fig 6.7.



**Fig. 6.7 Font Dialog Box**

---

### 6.6.3.1 FONT DIALOG BOX PROPERTIES

---

Font Dialog Box properties are shown in Table 6.7 and there are fourteen properties.

**Table 6.7 Font Dialog Box Properties**

S.N.	Property	Description
1	AllowSimulations	Gets or sets a value indicating whether the dialog box allows graphics device interface (GDI) font simulations.
2	AllowVectorFonts	Gets or sets a value indicating whether the dialog box allows vector font selections.
3	AllowVerticalFonts	Gets or sets a value indicating whether the dialog box displays both vertical and horizontal fonts, or only horizontal fonts.
4	Color	Gets or sets the selected font color.
5	FixedPitchOnly	Gets or sets a value indicating whether the dialog box allows only the selection of fixed-

		pitch fonts.
6	Font	Gets or sets the selected font.
7	FontMustExist	Gets or sets a value indicating whether the dialog box specifies an error condition if the user attempts to select a font or style that does not exist.
8	MaxSize	Gets or sets the maximum point size a user can select.
9	MinSize	Gets or sets the minimum point size a user can select.
10	ScriptsOnly	Gets or sets a value indicating whether the dialog box allows selection of fonts for all non-OEM and Symbol character sets, as well as the ANSI character set.
11	ShowApply	Gets or sets a value indicating whether the dialog box contains an <b>Apply</b> button.
12	ShowColor	Gets or sets a value indicating whether the dialog box displays the color choice.
13	ShowEffects	Gets or sets a value indicating whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options.
14	ShowHelp	Gets or sets a value indicating whether the dialog box displays a Help button.

---

### 6.6.3.2 METHODS OF THE FONTDIALOG CONTROL

---

Methods of the FontDialog control are shown in Table 6.8 and there are three methods.

**Table 6.8 Font Dialog Box Properties**

<b>S.N.</b>	<b>Method Name</b>	<b>Description</b>
1	Reset	Resets all options to their default values.
2	RunDialog	When overridden in a derived class, specifies a common dialog box.
3	ShowDialog	Runs a common dialog box with a default owner.

**Example:** In this example, let's change the font and color of the text from a rich text control using the Font dialog box. Take the following steps:

1. Drag and drop a RichTextBox control, a Button control and a FontDialog control on the form.
2. Set the Text property of the button control to 'Change Font'.
3. Set the ShowColor property of the FontDialog control to True.
4. Double-click the Change Color button and modify the code of the Click event.

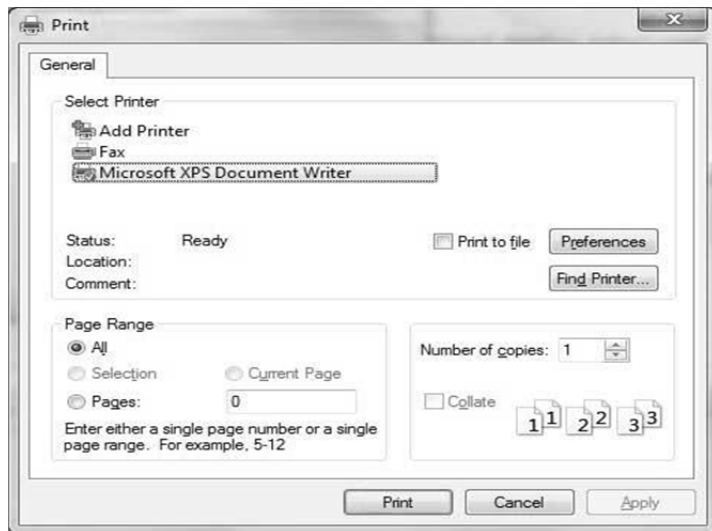
---

## **6.6.4 PRINT DIALOG BOX**

---

Print dialog box is also very important and it has several features (Fig.6.8). There are various other controls related to printing of documents. Let us have a brief look at these controls and their purpose. These other controls are:

- The **PrintDocument** control- it provides support for actual events and operations of printing in Visual Basic and sets the properties for printing.
- The **PrinterSettings** control- it is used to configure how a document is printed by specifying the printer.
- The **PageSetUpDialog** control- it allows the user to specify page-related print settings including page orientation, paper size and margin size.
- The **PrintPreviewControl** control- it represents the raw preview part of print previewing from a Windows Forms application, without any dialog boxes or buttons.



**Fig. 6.8 Print Dialog Box**

### 6.6.4.1 PRINT DIALOG BOX PROPERTIES

Print Dialog Box properties are shown in Table 6.9 and there are nine properties.

**Table 6.9 Print Dialog Box Properties**

S.No.	Property	Description
1	AllowCurrentPage	Gets or sets a value indicating whether the <i>Current Page</i> option button is displayed.
2	AllowPrintToFile	Gets or sets a value indicating whether the <i>Print to file</i> check box is enabled.
3	AllowSelection	Gets or sets a value indicating whether the <i>Selection</i> option button is enabled.
4	AllowSomePages	Gets or sets a value indicating whether the <i>Pages</i> option button is enabled.
5	Document	Gets or sets a value indicating the <i>PrintDocument</i> used to obtain <i>PrinterSettings</i> .

6	PrinterSettings	Gets or sets the printer settings the dialog box modifies.
7	PrintToFile	Gets or sets a value indicating whether the <i>Print to file</i> check box is selected.
8	ShowHelp	Gets or sets a value indicating whether the <i>Help</i> button is displayed.
9	ShowNetwork	Gets or sets a value indicating whether the <i>Network</i> button is displayed.

---

#### 6.6.4.2 PRINT DIALOG BOX METHODS

---

Print Dialog Box methods are shown in Table 6.10 and there are three properties.

**Table 6.10 Print Dialog Box Properties**

S.No.	Method Name	Description
1	Reset	Resets all options to their default values.
2	RunDialog	When overridden in a derived class, specifies a common dialog box.
3	ShowDialog	Runs a common dialog box with a default owner.

**Example:** In this example, let us see how to show a Print dialog box in a form. Take the following steps:

1. Add a PrintDocument control, a PrintDialog control and a Button control on the form. The PrintDocument and the PrintDialog controls are found on the Print category of the controls toolbox.
2. Change the text of the button to 'Print'.
3. Double-click the Print button and modify the code of the Click even

---

## 6.7 VISUAL BASIC AND INTERNET

---

In order to create the web browser, *Microsoft Internet Control* is



required. This control is not available in default VB6 IDE, you have to add it from the components window. To add this control, press Ctrl+T to open up the components window and select Microsoft Internet Control. After selecting the control, you will see the control appears in the toolbox as a small globe. To insert the Microsoft Internet Control into the form, just drag the globe into the form and a white rectangle will appear in the form. You can resize this control as you wish. This control is given the default name *WebBrowser1*.

To design the interface, you need to insert one combo box which will be used to display the URLs. In addition, you can insert a few images which will function as command buttons for the user to navigate the web; they are the *Go* command, the *Back* command, the *Forward* command, the *Refresh* command and the *Home* command. You can actually put in the command buttons instead of the images, but using images will definitely improve the appearance of the browser.

The procedures for all the commands are relatively easy to code. There are many methods, events, and properties associated with the web browser but you need to know just a few of them to come up with a functional Internet browser. The method navigate is to go the website specified by its Uniform Resource Locator(URL). The syntax is `WebBrowser1.Navigate ("URL")`.

---

### 6.7.1 THE CODE

---

```
Private Sub Form_Load()  
WebBrowser1.Navigate ("http://www.gkv.ac.in")  
End Sub
```

In order to show the URL in the combo box and also to display the page title at the form caption after the page is completely downloaded, we use the following statements:

```
Private Sub  
WebBrowser1_DocumentComplete (ByVal pDisp As Object, URL As  
Variant)  
Combo1.Text = URL  
Form1.Caption = WebBrowser1.LocationName  
Combo1.AddItem URL  
End Sub
```

The following procedure will tell the user to wait while the page is loading.

```
Private Sub  
WebBrowser1_DownloadBegin ()  
Combo1.Text = "Page loading, please wait"  
End Sub
```

As the time passed Internet applications need more features and methods. Hence Visual Basic 6 faced some problems too, and hence new things like complete object oriented features were included in VB.NET. The last version of VB, VB 6, was released in 1998, but has since been replaced by VB.NET, Visual Basic for applications (VBA) and Visual Studio .NET. VBA and Visual Studio are the two frameworks most commonly used so far.

### **CHECK YOUR PROGRESS**

- What do you understand by common dialog box?
- Give the name of some common dialog boxes.
- Discuss the properties and methods of file open dialog box.

---

## **6.8 SUMMARY**

---

A *dialog box* is simply a form in a program that contains input controls designed to receive information. To make your programming faster, VB includes an ActiveX control, named `CommonDialog`. With this control, it can easily be displayed six standard dialog boxes in the programs.

A Windows form can be displayed in one of two modes, *modal* and *non-modal (modeless)*. When a form is non-modal it means that other forms in the other forms in the application remain accessible to the user (in that they can still click on controls or use the keyboard in other forms). When a form is modal, as soon as it is displayed all other forms in the application are disabled until the modal dialog is dismissed by the user. Modal forms are typically used when the user is required to complete a task before proceeding to another part of the application.

The Common Dialog control provides a standard set of dialog boxes for operations such as opening, saving, and printing files, as well as selecting colors and fonts and displaying help. A particular dialog box is displayed by using one of the six "Show..." methods of the Common Dialog control: *ShowOpen*, *ShowSave*, *ShowPrinter*, *ShowColor*, *ShowFont*, or *ShowHelp*.

The Common Dialog control not an intrinsic control; rather, it is an "Active X" control that must be added to the toolbox via the *Components* dialog box. This dialog box is accessed via the Project menu, Components item. Once "Microsoft Common Dialog Control 6.0" is checked and clicked OK, the control is added to the toolbox. Thereafter double-click it to make it appear on the form, as you would with any other control. The Common Dialog control is not visible at run-time.

To create a web browser, Internet Control is needed in VB applications. This control is not available in default VB6 IDE, it is to be added from the components window. To add this control, Ctrl+T can be pressed to open up the components window and select Microsoft Internet Control. After selected

the control, the control appears in the toolbox as a small globe. To insert the Microsoft Internet Control into the form, the globe is dragged into the form and a white rectangle will appear in the form. This control can be resized as per the wish of the user. This control is given the default name *WebBrowser1*.

---

## 6.9 TERMINAL QUESTIONS

---

1. What do you understand by dialog box? Explain.
2. Explain modal dialog box using example.
3. Write a short note on modeless dialog box with example.
4. Compare modal and modeless dialog box.
5. What do you understand by common dialog control? Explain briefly.
6. Explain font dialog box using its important properties and methods.
7. What do you understand by a print dialog box? Explain its properties and methods.
8. Write a short note on the use of Internet in Visual Basic.





॥ सरस्वती नः सुभगा मयस्करत् ॥

Uttar Pradesh Rajarshi Tandon  
Open University

Bachelor in Computer  
Application

**BCA-118**

**Windows Programming**

**BLOCK**

**3**

**WORKING WITH GRAPHICS**

---

**UNIT-7**

**Document View Architecture**

---

**UNIT-8**

**Graphics and Multimedia**

---

---

## Course Design Committee

---

**Prof. Ashutosh Gupta**

Director

School of Science, UPRTOU Prayagraj

**Prof. Suneeta Agarwal**

Dept. of Computer Science & Engineering

Motilal Nehru National Institute of Technology, Allahabad, Prayagraj

**Dr. Upendra Nath Tripathi**

Associate Professor

Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

**Dr. Ashish Khare**

Associate Professor

Dept. of Computer Science, University of Allahabad, Prayagraj

**Ms. Marisha**

Assistant Professor (Computer Science)

School of Science, UPRTOU Prayagraj

**Mr. Manoj Kumar Balwant**

Assistant Professor (Computer Science)

School of Sciences, UPRTOU Prayagraj

---

## Course Preparation Committee

---

**Dr. Krishan Kumar**

**Author**

Assistant Professor

Department of Computer Science,

Gurukula Kangri Vishwavidyalaya Haridwar (UK)

**Dr. Brajesh Kumar**

**Editor**

Associate Professor, Dept. of CS & IT

M.J.P Rohilkahand University, Bareilly, Uttar Pradesh

**Prof. Ashutosh Gupta**

**Director (In-Charge)**

School of Computer & Information Sciences

UPRTOU Prayagraj

**Mr. Manoj Kumar Balwant**

**Coordinator**

Assistant Professor (computer science)

School of Sciences, UPRTOU Prayagraj

---

©UPRTOU, Prayagraj - 2020

ISBN :

---

©All Rights are reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the **Uttar Pradesh Rajarshi Tondon Open University, Prayagraj.**

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.

**Printed By :** Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

---

## BLOCK INTRODUCTION

---

Block 3 includes two units 7 and 8. Unit 7 focusses on document view architecture and Unit 8 is related with graphics and multimedia. At the end of this block one would be able to know about some of the following crucial concepts

- MFC
- SDI
- MDI
- VC++ resources
- Menu and toolbars
- Multithreading
- Clipboards

*Unit 7* describes about the document view architecture concepts using MFC. MFC plays a very important role in the development of this architecture. It explains about the SDI and MDI. As we know MFC are the foundation classes which provides a good platform for the windows programming. It explains about the MFC library. Moreover, this unit describes about the serialization and separating documents from the views. Furthermore, it gives the details of Visual C++ Resources: Application Wizard, Accelerators and Menus, Toolbars.

*Unit 8* mainly focusses on the working with graphics and multimedia. Graphics and multimedia provides a good interface for all the applications. Without an interface an application leaves a bad impression though it may be very robust. Hence this unit basically intends with consoles, multitasking process and threads, drawing graphics in Windows, setting colors, drawing text, lines, ellipses, arcs, circles, plotting points; filling figures with colors and patterns, Clipboard Drag and Drops, using clipboards to transfer images between applications.





---

# UNIT-7 DOCUMENT VIEW ARCHITECTURE

---

## Structure

- 7.0 Introduction
- 7.1 Objectives
- 7.2 Microsoft Foundation Class
- 7.3 View Document Architecture Using MFC
- 7.4 Serialization
- 7.5 Separating documents from view
- 7.6 Visual C++ Resources
- 7.7 Application Wizard
- 7.8 Accelerators
- 7.9 Menus
- 7.10 Toolbars
- 7.11 Summary
- 7.12 Terminal Questions

---

## 7.0 INTRODUCTION

---

More complex Microsoft Foundation Class (MFC) applications use the Document-View architecture, which decouples the user interface (the view) from the application data and logic (the document). The calculator's number stack is contained in its document component. This is a common pattern. Often the data of an MFC application will consist of many objects. In these cases, the document becomes the manager of these objects, using a container of some type to store and organize them. Lists, arrays, stacks, queues, tables, and sets are familiar examples of containers. MFC provides templates for some of these containers.

The MFC library provides a set of functions, constants, data types, and classes to simplify creating applications for the Microsoft Windows operating systems. In this unit, you will learn all about how to start and create Windows-based applications using MFC. To gain the advantage one need to be familiar with programming for Windows. He/she also need to know the basics of programming in C++ and understand the fundamentals of object-oriented programming. By default, the MFC Application Wizard creates an application skeleton with a document class and a view class. MFC separates data management into these two classes. The document stores the data and manages printing the data and coordinates updating multiple views of the data. The view displays the data and manages user interaction

with it, including selection and editing.

In this model, an MFC document object reads and writes data to persistent storage. The document may also provide an interface to the data wherever it resides (such as in a database). A separate view object manages data display, from rendering the data in a window to user selection and editing of data. The view obtains display data from the document and communicates back to the document any data changes.

---

## 7.1 OBJECTIVES

---

At the end of this unit you will come to know about the following

- Microsoft Foundation Classes (MFC)
- View Document Architecture
- Serialization
- Separating documents from view
- Visual C++ Resources
- Application Wizard
- Accelerators
- Menus
- Toolbars

---

## 7.2 MICROSOFT FOUNDATION CLASS

---

The Microsoft Foundation Class (MFC) library provides a set of functions, constants, data types, and classes to simplify creating applications for the Microsoft Windows operating systems. In this tutorial, you will learn all about how to start and create windows based applications using MFC.

---

### 7.2.1 PREREQUISITES

---

It is assumed that you must know the following things before knowing MFC:

- A little about programming for Windows.
- The basics of programming in C++.
- Understand the fundamentals of object-oriented programming.

---

### 7.2.2 What is MFC?

---

The MFC is an "application framework" for programming in Microsoft Windows. It provides much of the code, which are required for the following:

- Managing Windows.
- Menus and dialog boxes.

- Performing basic input/output.
- Storing collections of data objects, etc.

You can easily extend or override the basic functionality of the MFC framework in C++ applications by adding your application-specific code into MFC framework.

---

### 7.2.3 MFC FRAMEWORK

---

- The MFC framework provides a set of reusable classes designed to simplify Windows programming.
- MFC provides classes for many basic objects, such as strings, files, and collections that are used in everyday programming.
- It also provides classes for common Windows APIs and data structures, such as windows, controls, and device contexts.
- The framework also provides a solid foundation for more advanced features, such as ActiveX control and document view processing.
- In addition, MFC provides an application framework, including the classes that make up the application architecture hierarchy.

---

### 7.2.4 SIGNIFICANCE OF MFC

---

The MFC framework is a powerful approach that lets you build upon the work of expert programmers for Windows. MFC framework has the following advantages:

- It shortens development time.
- It makes code more portable.
- It provides tremendous support without reducing programming freedom and flexibility.
- It gives easy access to "hard to program" user-interface elements and technologies.
- MFC simplifies database programming through Data Access Objects (DAO) and Open Database Connectivity (ODBC), and network programming through Windows Sockets.

---

## 7.3 DOCUMENT VIEW ARCHITECTURE USING MFC

---

The *Document/View architecture* is the foundation used to create applications based on the Microsoft Foundation Classes library. It allows you to make the different parts that compose a computer program including what the user sees as part of the application and the document a user would work on. This is done through a combination of separate classes that work as an ensemble. You

can easily extend or override the basic functionality of the MFC framework in your C++ applications by adding your application-specific code into MFC framework. The parts that compose the Document/View architecture are a *frame*, one or more *documents*, and the *view*. These three entities are put together, to make up a usable application.

**View-** A *view* is the platform the user is working on to do his/ her job. To let the user, do anything on an application, you must provide a view, which is an object based on the *CView* class. You can either directly use one of the classes derived from *CView* or you can derive your own custom class from *CView* or one of its child classes.

**Document-** A *document* is similar to a bucket. For a computer application, a document holds the user's data. To create the document part of this architecture, you must derive an object from the *CDocument* class.

**Frame-** As the name suggests, a *frame* is a combination of the building blocks, the structure, and the borders of an item. A frame gives "physical" presence to a window. It also defines the location of an object with regards to the Windows desktop.

---

### 7.3.1 SINGLE DOCUMENT INTERFACE

---

The expression *Single Document Interface* or SDI refers to a document that can present only one view to the user. This means that the application cannot display more than one document at a time. If you want to view another type of document of the current application, you must create another instance of the application. Notepad and WordPad are good examples of SDI applications.

Now, let us look into a simple example of single document interface or SDI by creating a new MFC dialog based application.

**Step 1-** Let us create a new MFC Application *MFCSDIDemo* with the settings mentioned in Fig. 7.1.

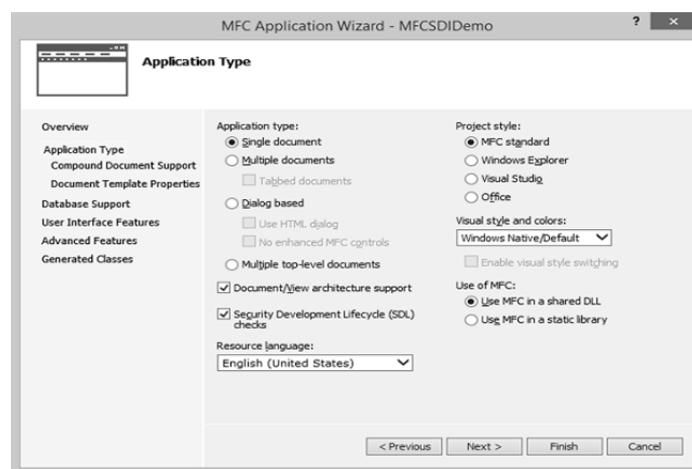


Fig. 7.1 MFC Application wizard

**Step 2**– Select Single document from the Application type and MFC standard from Project Style.

**Step 3**– Click Finish to Continue.

**Step 4**– Once the project is created, run the application and you will see the output shown in Fig. 7.2.



**Fig. 7.2 MFCSDI Demo**

---

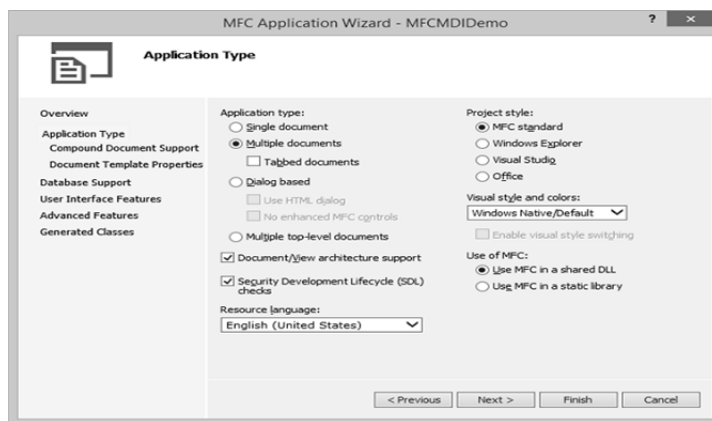
### 7.3.2 MULTIPLE DOCUMENT INTERFACE

---

An application is referred to as a *Multiple Document Interface*, or MDI, if the user can open more than one document in the application without closing it. To provide this functionality, the application provides a parent frame that acts as the main frame of the computer program. Inside this frame, the application allows creating views with individual frames, making each view distinct from the other.

Let us look into a simple example of multiple document interface or MDI by creating a new MFC dialog based application.

**Step 1**– Let us create a new MFC Application *MFCMDIDemo* with below mentioned settings (Fig. 7.3).

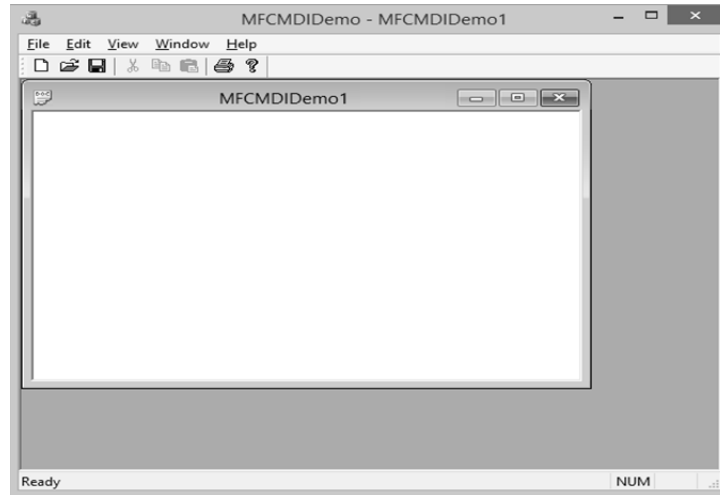


**Fig. 7.3 MFCMDI Demo-1**

**Step 2**– Select Multiple document from the Application type and MFC standard from Project Style.

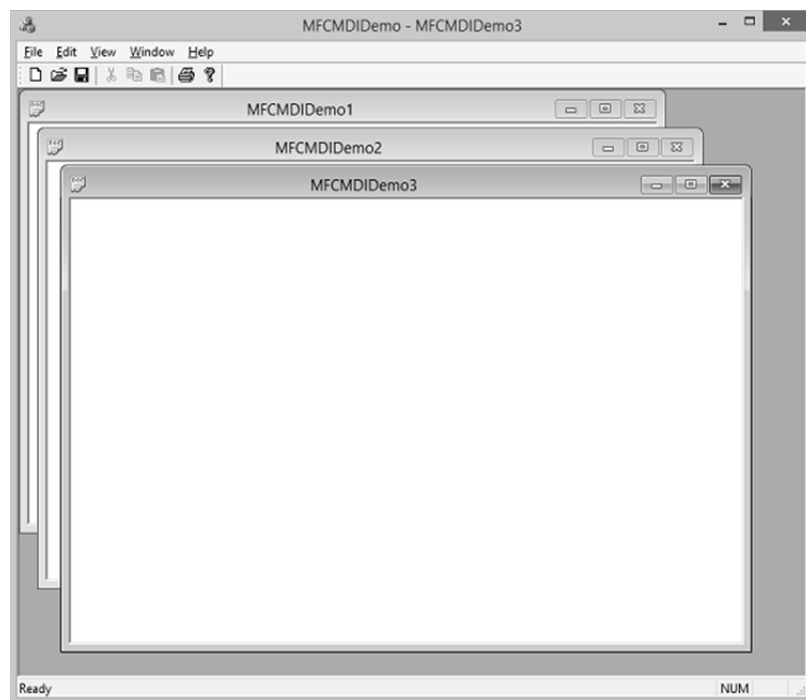
**Step 3**– Click Finish to Continue.

**Step 4**– Once the project is created, run the application and you will see the following output shown in Fig. 7.4.



**Fig. 7.4 MFCMDI Demo-2**

**Step 5** – When you click on File → New menu option, it will create another child window as shown in the following snapshot (Fig. 7.5).



**Fig. 7.5 MFCMDI Demo-3**

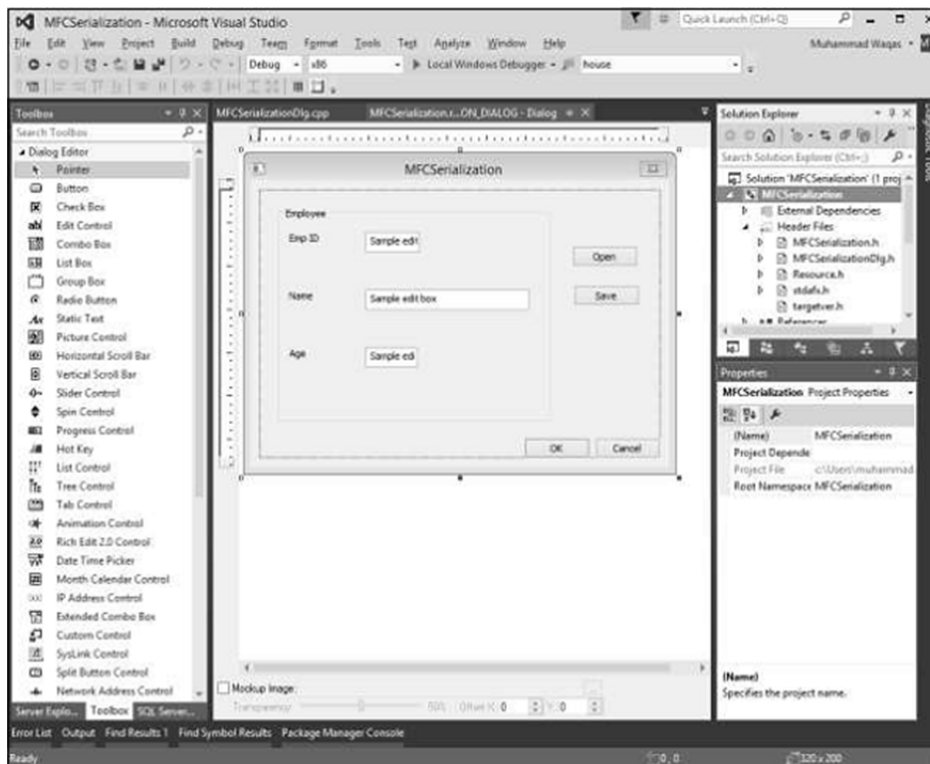
**Step 6**– In Multiple Document Interface (MDI) applications, there is one main frame per application. In this case, a `CMDIFrameWnd`, and one `CMDIChildWnd` derived child frame for each document.

## 7.4 SERIALIZATION

*Serialization* is the process of writing or reading an object to or from a persistent storage medium such as a disk file. Serialization is ideal for situations where it is desired to maintain the state of structured data (such as C++ classes or structures) during or after the execution of a program. When performing file processing, the values are typically of primitive types (char, short, int, float, or double). In the same way, we can individually save many values, one at a time. This technique doesn't include an object created from (as a variable of) a class. The MFC library has a high level of support for serialization. It starts with the *CObject* class that is the ancestor to most MFC classes, which is equipped with a *Serialize()* member function.

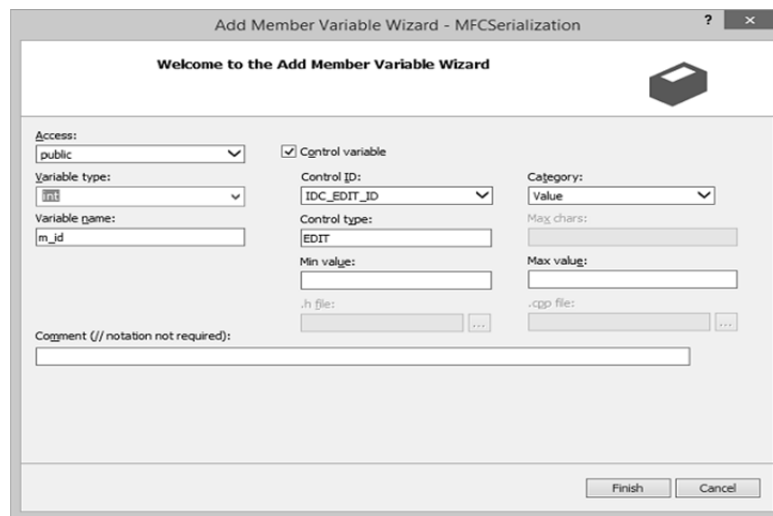
Now, let us look into a simple example by creating a new MFC project.

**Step 1**– Remove the TODO line and design your dialog box as shown in the Fig. 7.6 below.



**Fig. 7.6** Design a dialog box

**Step 2**– Add value variables for all the edit controls. For Emp ID and Age mentioned, the value type is an integer as shown in the Fig. 7.7 below.



**Fig. 7.7 Adding member variable wizard**

**Step 3**– Add the event handler for both the buttons.

**Step 4**– Let us now add a simple Employee class, which we need to serialize. Here is the declaration of Employee class in header file.

```
class CEmployee : public CObject
{
public:
    int empID;
    CString empName;
    int age;
    CEmployee(void);
    ~CEmployee(void);

private:
public:
    void Serialize(CArchive& ar);
    DECLARE_SERIAL(CEmployee);
};
```

**Step 5**– Here is the definition of Employee class in source (\*.cpp) file.

```
IMPLEMENT_SERIAL(CEmployee, CObject, 0)
CEmployee::CEmployee(void) { }
CEmployee::~~CEmployee(void) { }
```



```

void CEmployee::Serialize(CArchive& ar) {
    CObject::Serialize(ar);
    if (ar.IsStoring())
        ar << empID << empName << age;
    else
        ar >> empID >> empName >> age;
}

```

**Step 6**– Here is the implementation of Save button event handler.

```

void CMFCSerializationDlg::OnBnClickedButtonSave() {
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    CEmployee employee;
    CFile file;
    file.Open(L"EmployeeInfo.hse",CFile::modeCreate|CFile::modeWrite);
    CArchive ar(&file, CArchive::store);
    employee.empID = m_id;
    employee.empName = m_strName;
    employee.age = m_age;
    employee.Serialize(ar);
    ar.Close();
}

```

**Step 7**– Here is the implementation of Open button event handler.

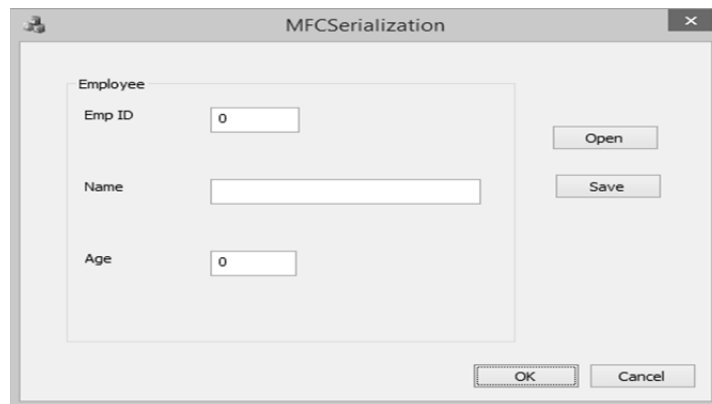
```

void CMFCSerializationDlg::OnBnClickedButtonOpen() {
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    CFile file;
    file.Open(L"EmployeeInfo.hse", CFile::modeRead);
    CArchive ar(&file, CArchive::load);
    CEmployee employee;

```

```
employee.Serialize(ar);
m_id = employee.empID;
m_strName = employee.empName;
m_age = employee.age;
ar.Close();
file.Close();
UpdateData(FALSE);
}
```

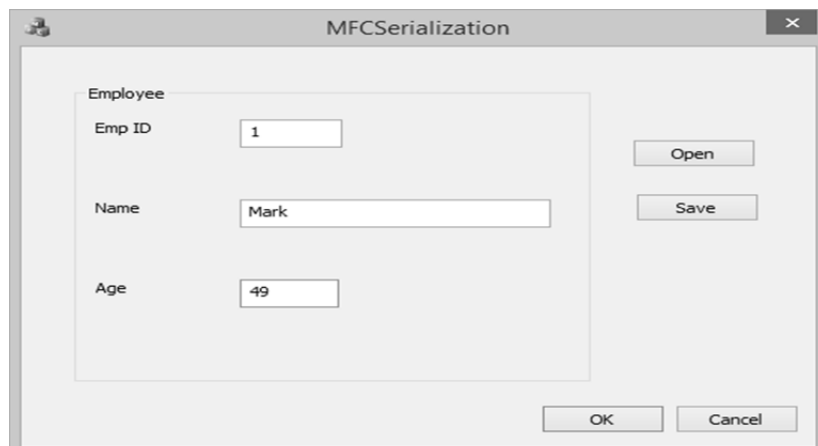
**Step 8**– When the above code is compiled and executed, you will see the following output (Fig. 7.8).



The screenshot shows a window titled "MFCSerialization". Inside the window, there is a group box labeled "Employee". Within this group box, there are three input fields: "Emp ID" with the value "0", "Name" which is empty, and "Age" with the value "0". To the right of the group box, there are two buttons: "Open" and "Save". At the bottom of the window, there are two buttons: "OK" and "Cancel".

**Fig. 7.8 MFCSerialization- output**

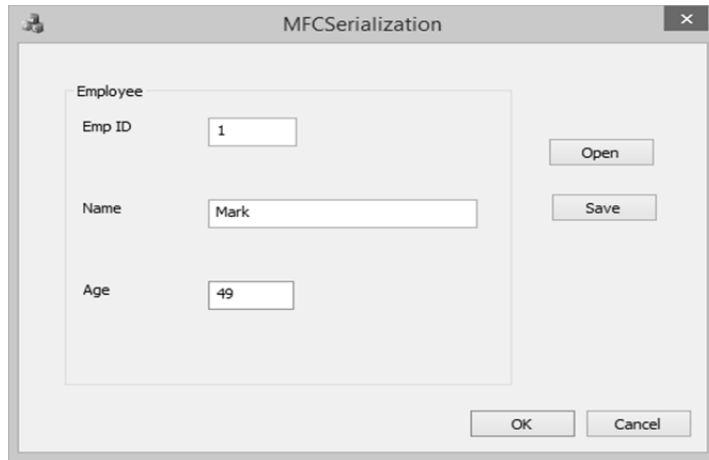
**Step 9**– Enter the info in all the fields shown in the Fig. 7.9 and click Save and close this program.



The screenshot shows the same "MFCSerialization" window as in Fig. 7.8, but now the input fields are filled with data. The "Emp ID" field contains "1", the "Name" field contains "Mark", and the "Age" field contains "49". The "Open" and "Save" buttons are still present to the right, and the "OK" and "Cancel" buttons are at the bottom.

**Fig. 7.9 MFCSerialization- filling the info**

**Step 10**– It will save the data. Run the application again and click open. It will load the Employee information (Fig. 7.10).



**Fig. 7.10 MFCSerialization- loading the employee info**

### **CHECK YOUR PROGRESS**

- What do you understand by Microsoft Foundation Classes (MFC)?
- Describe the term view document architecture.
- Discuss the term serilization.

---

## **7.5 SEPARATING DOCUMENTS FROM VIEW**

---

This topic is intended with the interactions between documents and view. It is really routing that takes place between the document object and view object. Basically the data of the user is maintained by the document and the way it is presented to the user by means of view. The *CFormView* class is used as the base class for the users' views.

---

### **7.5.1 DOCUMENT-VIEW INTERACTION FUNCTIONS**

---

You already know that the *document object holds the data* and that the *view object displays the data and allows editing*. An SDI application has a document class derived from *CDocument*, and it has one or more view classes, each ultimately derived from *CView*. A complex handshaking process takes place among the document, the view, and the rest of the application framework. To understand this process, you need to know about five important member functions in the document and view classes. Two are non-virtual base class functions that you call in your derived classes; three are virtual functions that you often override in your derived classes. Let's look at these functions one at a time.

---

### 7.5.1.1 THE CVIEW::GETDOCUMENT FUNCTION

---

A view object has one and only one associated document object. The *GetDocument()* function allows an application to navigate from a view to its document. Suppose a view object gets a message that the user has entered new data into an edit control. The view must tell the document object to update its internal data accordingly. The *GetDocument()* function provides the document pointer that can be used to access document class member functions or public data members.

The *CDocument::GetNextView* function navigates from the document to the view, but because a document can have more than one view, it's necessary to call this member function once for each view, inside a loop. You'll seldom call *GetNextView()* because the application framework provides a better method of iterating through a document's views.

When AppWizard generates a derived *CView* class, it creates a special type-safe version of the *GetDocument()* function that returns not a *CDocument()* pointer but a pointer to an object of your derived class. This function is an inline function, and it looks something like:

```
CMyDoc* GetDocument()
{
    return (CMyDoc*)m_pDocument;
}
```

When the compiler sees a call to *GetDocument()* in your view class code, it uses the derived class version instead of the *CDocument()* version, so you do not have to cast the returned pointer to your derived document class. Because the *CView::GetDocument* function is not a virtual function, a statement such as:

```
pView->GetDocument(); // pView is declared CView*
```

Calls the base class *GetDocument()* function and thus returns a pointer to a *CDocument* object.

---

### 7.5.1.2 THE CDOCUMENT::UPDATEALLVIEWS FUNCTION

---

If the document data changes for any reason, all views must be notified so that they can update their representations of that data. If *UpdateAllViews()* is called from a member function of a derived document class, its first parameter, *pSender*, is *NULL*. If *UpdateAllViews()* is called from a member function of a derived view class, set the *pSender* parameter to the current view, like this:

```
GetDocument () ->UpdateAllViews(this);
```

The non-null parameter prevents the application framework from notifying the current view. The assumption here is that the current view has already updated itself. The function has optional hint parameters that can be used to give view-specific and application-dependent information about which parts of the view to update. This is an advanced use of the function. How exactly is a view notified

when *UpdateAllViews()* gets called? Take a look at the next function, *OnUpdate()*.

---

### 7.5.1.3 THE CVIEW::ONUPDATE FUNCTION

---

This virtual function is called by the application framework in response to your application's call to the *CDocument::UpdateAllViews* function. You can, of course, call it directly within your derived *CView* class. Typically, your derived view class's *OnUpdate()* function accesses the document, gets the document's data, and then updates the view's data members or controls to reflect the changes. Alternatively, *OnUpdate()* can invalidate a portion of the view, causing the view's *OnDraw()* function to use document data to draw in the window. The *OnUpdate()* function might look something like this:

```
void CMyView::OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint)
{
    CMyDocument* pMyDoc = GetDocument();
    CString lastName = pMyDoc->GetLastName();
    // m_pNameStatic is a CMyView data member
    m_pNameStatic->SetWindowText(lastName);
}
```

The hint information is passed through directly from the call to *UpdateAllViews()*. The default *OnUpdate()* implementation invalidates the entire window rectangle. In your overridden version, you can choose to define a smaller invalid rectangle as specified by the hint information. If the *CDocument()* function *UpdateAllViews()* is called with the *pSender* parameter pointing to a specific view object, *OnUpdate()* is called for all the document's views except the specified view.

---

### 7.5.1.4 THE CVIEW::ONINITIALUPDATE FUNCTION

---

This virtual *CView* function is called when the application starts, when the user chooses *New* from the *File* menu, and when the user chooses *Open* from the *File* menu. The *CView* base class version of *OnInitialUpdate()* does nothing but call *OnUpdate()*. If you override *OnInitialUpdate()* in your derived view class, be sure that the view class calls the base class's *OnInitialUpdate()* function or the derived class's *OnUpdate()* function. You can use your derived class's *OnInitialUpdate()* function to initialize your view object. When the application starts, the application framework calls *OnInitialUpdate()* immediately after *OnCreate()* (if you've mapped *OnCreate()* in your view class). *OnCreate()* is called once, but *OnInitialUpdate()* can be called many times.

---

### 7.5.1.5 THE CDOCUMENT::ONNEWDOCUMENT FUNCTION

---

The framework calls this virtual function after a document object is first

constructed and when the user chooses *New* from the *File* menu in an SDI application. This is a good place to set the initial values of your document's data members. AppWizard generates an overridden *OnNewDocument()* function in your derived document class. Be sure to retain the call to the base class function.

---

## 7.5.2 THE SIMPLEST DOCUMENT - VIEW APPLICATION

---

Suppose you don't need multiple views of your document but you plan to take advantage of the application framework's file support. In this case, you can forget about the *UpdateAllViews()* and *OnUpdate()* functions. Simply follow the below steps when you develop the application:

1. In your derived document class header file (generated by AppWizard), declare your document's data members. These data members are the primary data storage for your application. You can make these data members public, or you can declare the derived view class a friend of the document class.
2. In your derived view class, override the *OnInitialUpdate()* virtual member function. The application framework calls this function after the document data has been initialized or read from disk. *OnInitialUpdate()* should update the view to reflect the current document data.
3. In your derived view class, let your window message handlers, command message handlers and your *OnDraw()* function read and update the document data members directly, using *GetDocument()* to access the document object.

The sequence of events for this simplified document-view environment is shown in Table 7.1

**Table 7.1 Sequence of events in document-view environment**

Sequence	Description
Application starts	CMyDocument object constructed CMyView object constructed View window created CMyView::OnCreate called (if mapped) CMyDocument::OnNewDocument called CMyView::OnInitialUpdate called View object initialized View window invalidated CMyView::OnDraw called
User edits data	CMyView functions update CMyDocument data members

User application	exits	CMyView object destroyed CMyDocument object destroyed
------------------	-------	--

---

## 7.6 VISUAL C++ RESOURCES

---

A *resource* is a text file that allows the compiler to manage objects such as pictures, sounds, mouse cursors, dialog boxes, etc. Microsoft Visual Studio makes creating a resource file particularly easy by providing the necessary tools in the same environment used to program. This means, you usually do not have to use an external application to create or configure a resource file. Some important features related to resources are:

- Resources are interface elements that provide information to the user.
- *Bitmaps, icons, toolbars, and cursors* are all resources.
- Some resources can be manipulated to perform an action such as selecting from a menu or entering data in dialog box.
- An application can use various resources that behave independently of each other, these resources are grouped into a text file that has the \*.rc extension.
- Most resources are created by selecting the desired one from the Add Resource dialog box. The Add Resource dialog box provides an extensive list of resources which can be used as per requirements.

**Note:-** *If you need something which is not available then you can add it manually to the \*.rc file before executing the program.*

---

### 7.6.1 IDENTIFIERS

---

An *identifier* is a symbol which is a constant integer whose name usually starts with ID. It consists of two parts – a text string (symbol name) mapped to an integer value (symbol value). Symbols provide a descriptive way of referring to resources and user-interface objects, both in your source code and while you're working with them in the resource editors. When you create a new resource or resource object, the *resource editors* provide a default name for the resource, for example, IDC\_DIALOG1, and assign a value to it. The name-plus-value definition is stored in the Resource.h file.

---

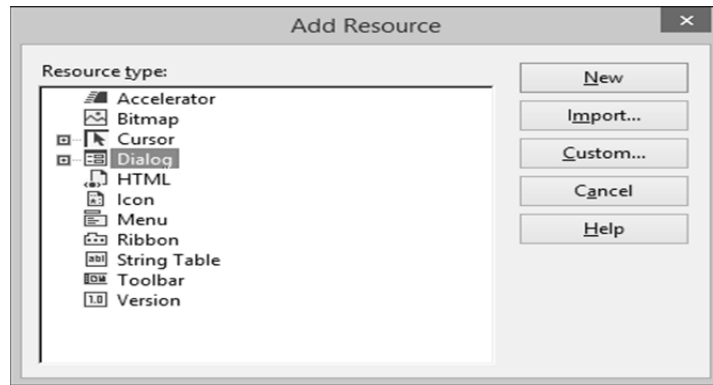
### 7.6.2 ICONS

---

An *icon* is a small picture used on a window which represents an application. It is used in two main scenarios – 1) On a Window's frame, it is displayed on the left side of the Window name on the title bar and 2) In Windows Explorer, on the Desktop, in My Computer, or in the Control Panel window.

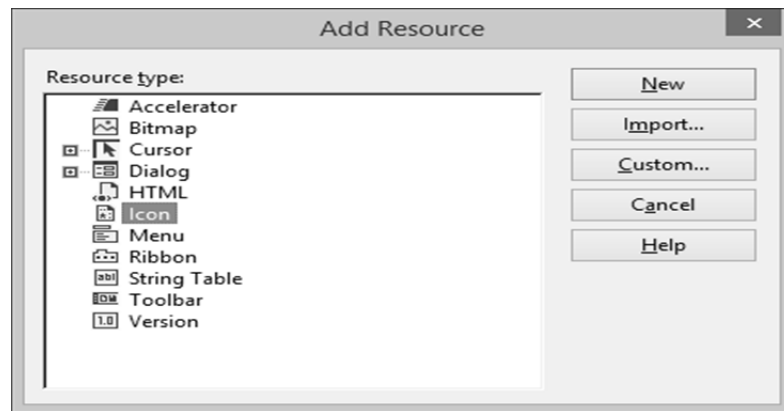
Own icons can be created by the steps given below –

**Step 1**– Right-click on your project and select Add → Resources, you will see the Add Resources dialog box (Fig. 7.11).

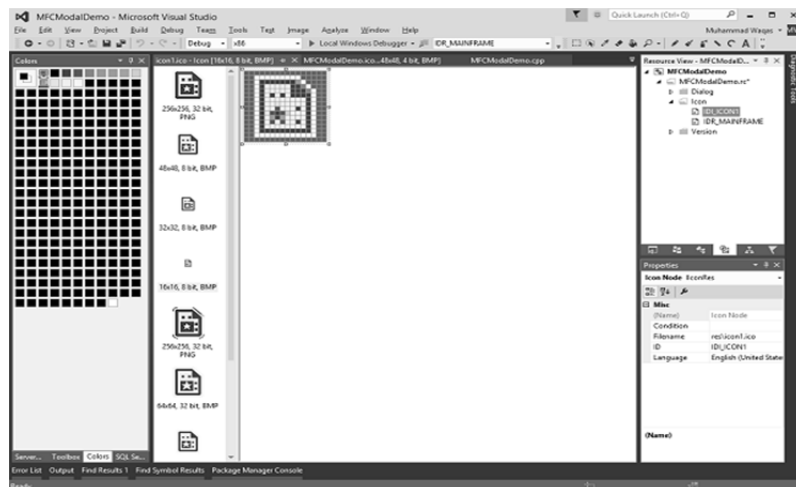


**Fig. 7.11** Adding resource dialog box

**Step 2**– Select Icon (Fig. 7.12) and click New button and you will see the icon shown in the Fig. 7.13.



**Fig. 7.12** Adding resource dialog box



**Fig. 7.13** Adding resource dialog box



**Step 3**– In Solution Explorer, go to Resource View and expand MFCModalDemo > Icon. You will see two icons. The IDR\_MAINFRAME is the default one and IDI\_ICON1 is the newly created icon.

**Step 4**– Right-click on the newly Created icon and select Properties.

**Step 5**– IDI\_ICON1 is the ID of this icon, now Let us change this ID to IDR\_MYICON.

**Step 6**– You can now change this icon in the designer as per your requirements.

---

## 7.7 APPLICATION WIZARD

---

The MFC Application Wizard generates an application that, when compiled, implements the basic features of a Windows executable (.exe) application. The MFC starter application includes C++ source (.cpp) files, resource (.rc) files, header (.h) files, and a project (.vcxproj) file. The code that is generated in these starter files is based on MFC. Depending on the options selected, the wizard creates additional files in the project. This wizard page describes the current application settings for the MFC application that you are creating. By default, the wizard creates a project as explained below.

**Application Type**- The project is created with tabbed MDI support; created using the Visual Studio project style and enables visual style switching. The project uses the Document/View Architecture, Unicode libraries, and MFC in a shared DLL.

**Compound Document Support**- The project provides no support for compound documents.

**Document Template Strings**- The project uses the project name for the default document template strings.

**Database Support** - The project provides no support for databases.

**User Interface Features**- The project implements standard Windows user interface features such as a system menu, a status bar, maximize and minimize boxes, an *About* box, a standard menu bar and docking toolbar, and child frames.

**Advanced Features**- The project supports printing and print preview. The project supports ActiveX controls. The project provides no support for Automation, MAPI, Windows Sockets, or Active Accessibility. The project supports an *Explorer* docking pane, an *Output* docking pane, and a *Properties* docking pane.

**Generated Classes**- The project's view class is derived from the CView Class. The project's application class is derived from the CWinAppEx Class. The project's document class is derived from the CDocument Class. The project's main frame class is derived from the CMDIFrameWndEx Class. The project's child frame class is derived from the CMDIChildWndEx Class.

**Note:** - *To change these default settings, click the appropriate tab title in the left column of the wizard and make the changes on the page that appears. After you create an MFC application project, you can add objects or controls to your project using Visual C++ code wizards.*

## CHECK YOUR PROGRESS

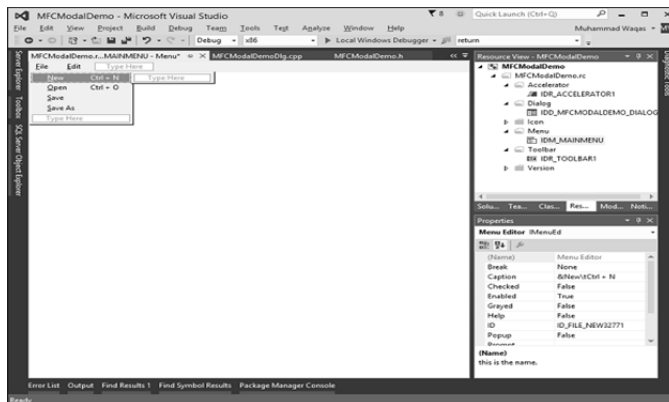
- What do you understand by separating documents?
- Describe the VC++ resources.
- Discuss about application wizard.

## 7.8 ACCELERATORS

A keyboard accelerator is basically a special way of translating a key or combination of keys pressed by the user into **WM\_COMMAND** messages like the ones your windows receive when the user clicks on a menu item. For example, most Windows programs have a menu option *Copy* under the *Edit* menu which copies the currently selected object (or objects) into the clipboard. Most programs also allow to perform the same operation by pressing the control key (Ctrl on most keyboards) and the C key at the same time. In fact, the message that is caused by the user clicking the Copy menu item and the message caused by the user pressing Ctrl+C is probably the same: a **WM\_COMMAND** message with an ID associated with the copy operation. Windows uses a table called an *accelerator table* to convert keystrokes by the user into messages to send to the program. These tables are resources which you can be created in a *resource script* and include in your programs to get similar functionality. Moreover, an *access key* is a letter that allows the user to perform a menu action faster by using the keyboard instead of the mouse. This is usually faster because the user would not need to position the mouse anywhere, which reduces the time it takes to perform the action.

### 7.8.1 SHORTCUT KEY

A shortcut key is a key or a combination of keys used by advanced users to perform an action that would otherwise be done on a menu item. Most shortcuts are a combination of the Ctrl key simultaneously pressed with a letter key. For example, Ctrl + N, Ctrl + O, or Ctrl + D. To create a shortcut, on the right side of the string that makes up a menu caption, right click on the menu item and select properties. In the Caption field type \t followed by the desired combination as shown below (Fig. 7.14) for the New menu option. Repeat the step for all menu options.



**Fig. 7.14 Adding resource dialog box**

## 7.8.2 ACCELERATOR TABLE

An Accelerator Table is a list of items where each item of the table combines an identifier, a shortcut key, and a constant number that specifies the kind of accelerator key. Applications often define keyboard shortcuts, such as CTRL+O for the File Open command. You could implement keyboard shortcuts by handling individual *WM\_KEYDOWN* messages, but accelerator tables provide a better solution that:

- Requires less coding.
- Consolidates all of your shortcuts into one data file.
- Supports localization into other languages.
- Enables shortcuts and menu commands to use the same application logic.

Just like the other resources, an accelerator table can be created manually in a .rc file. Follow the steps to create an accelerator table given below.

**Step 1**– To create an accelerator table (Fig. 7.15), right-click on \*.rc file in the solution explorer.



Fig. 7.15 Creating an accelerator table

**Step 2**– Select Accelerator and click New. It will display the interface shown in the Fig. 7.16.

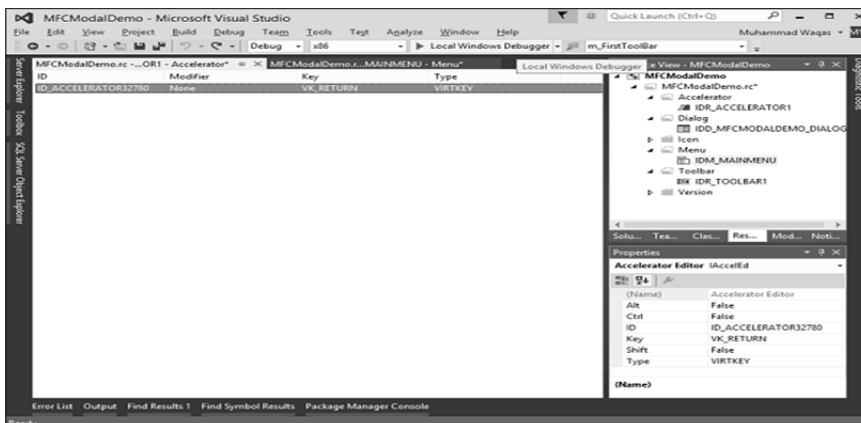
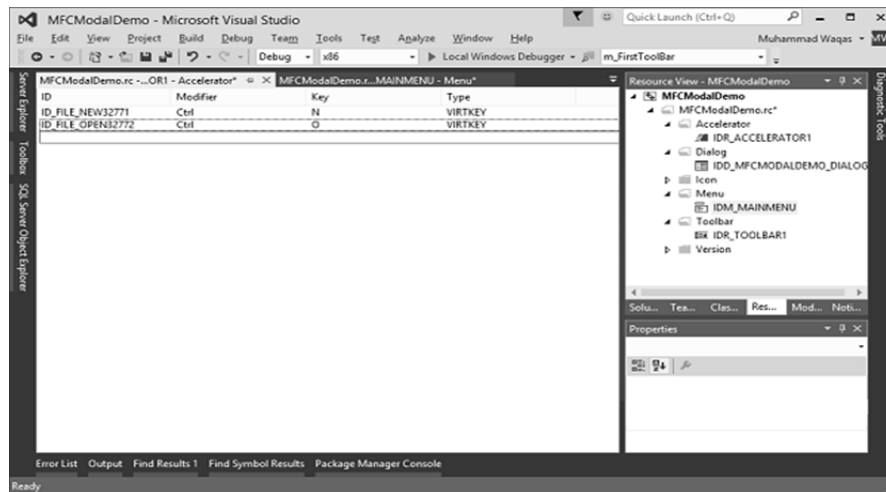


Fig. 7.16 Accelerator table showing combo box

**Step 3**– Click the arrow of the ID combo box and select menu Items as shown in Fig. 7.17.



**Fig. 7.17** Selecting menu Items

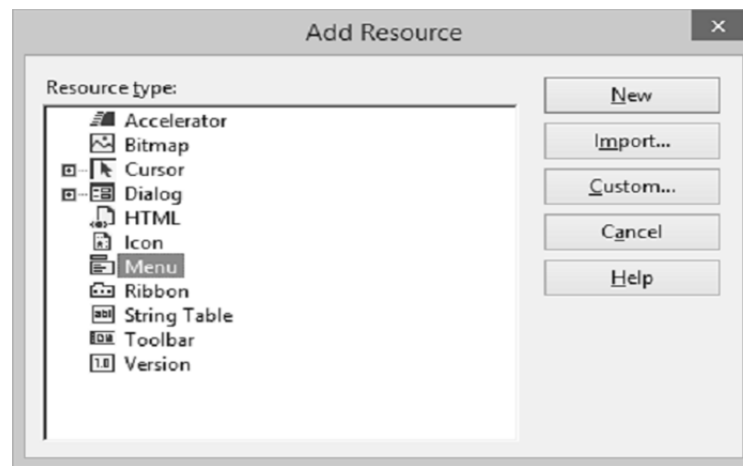
**Step 4**– Select Ctrl from the Modifier dropdown.

**Step 5**– Click the Key box and type the respective keys for both menu options.

## 7.9 MENUS

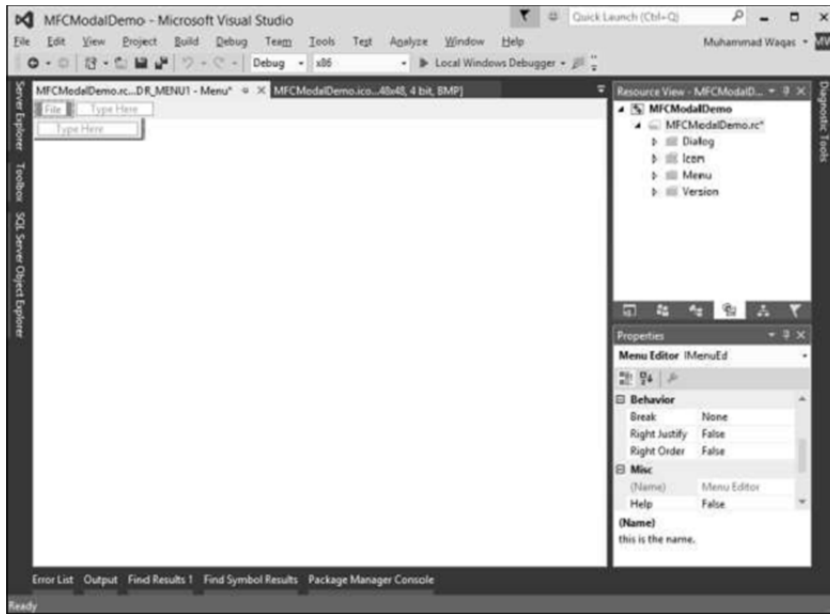
**Menus** allow to arrange commands in a logical and easy-to-find fashion. With the Menu editor, you can create and edit menus by working directly with a menu bar that closely resembles the one in your finished application. To create a menu, follow the steps given below

**Step 1**– Right-click on your project and select Add → Resources. You will see the Add Resources dialog box (Fig. 7.18).



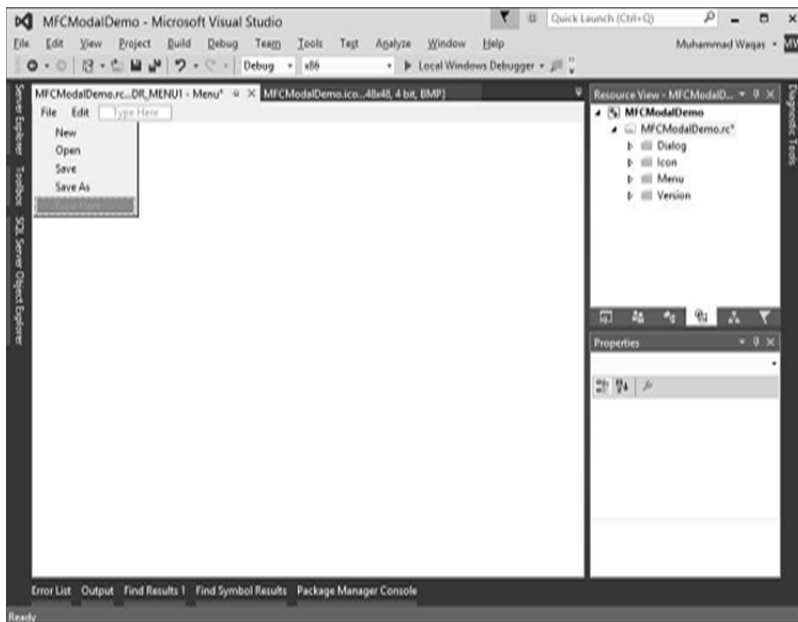
**Fig. 7.18** Adding resource dialog box

**Step 2**– Select Menu and click New. You will see the rectangle that contains "Type Here" on the menu bar (Fig. 7.19).



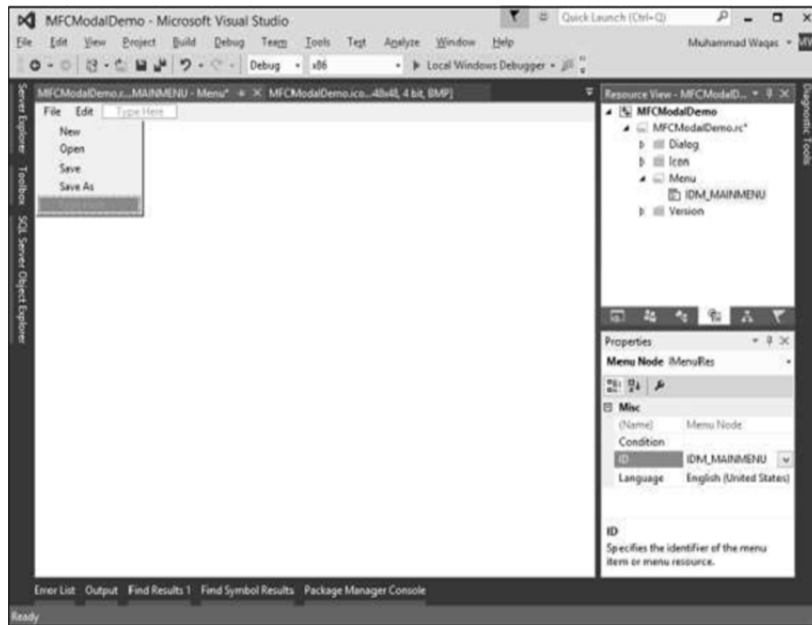
**Fig. 7.19** Type here box on the Menu bar

**Step 3**– Write some menu options like File, Edit, etc. as shown in the following snapshot (Fig. 7.20).



**Fig. 7.20** Menu options

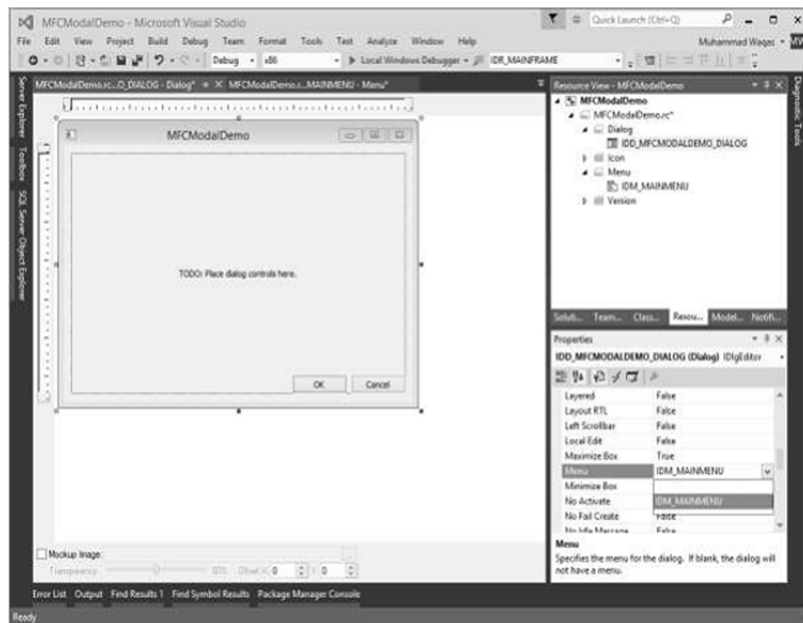
**Step 4**– If you expand the Menu folder in Resource View, you will see the Menu identifier IDR\_MENU1(7.21). Right-click on this identifier and change it to IDM\_MAINMENU.



**Fig. 7.21 Menu identifier**

**Step 5**– Save all the changes.

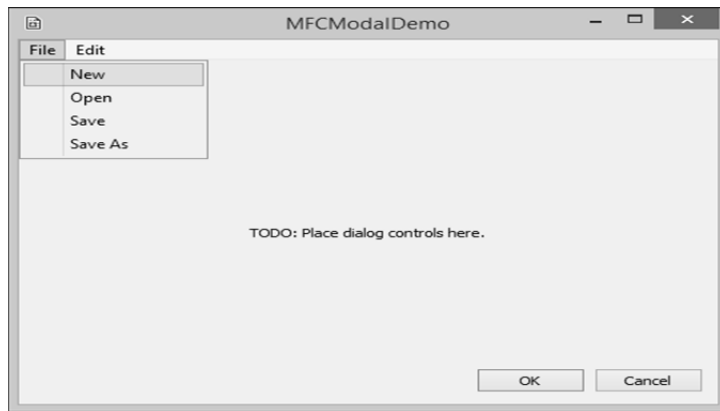
**Step 6**– We need to attach this menu to our dialog box. Expand your Dialog folder in Solution Explorer and double click on the dialog box identifier (Fig. 7.22).



**Fig. 7.22 Attaching menu to the dialog box**

**Step 7**– You will see the menu field in the Properties. Select the Menu identifier from the dropdown as shown above (Fig. 7.22).

**Step 8**– Run this application and you will see the following dialog box which also contains menu options (Fig. 7.23).



**Fig. 7.22** Output containing menu options

---

## 7.10 TOOLBAR

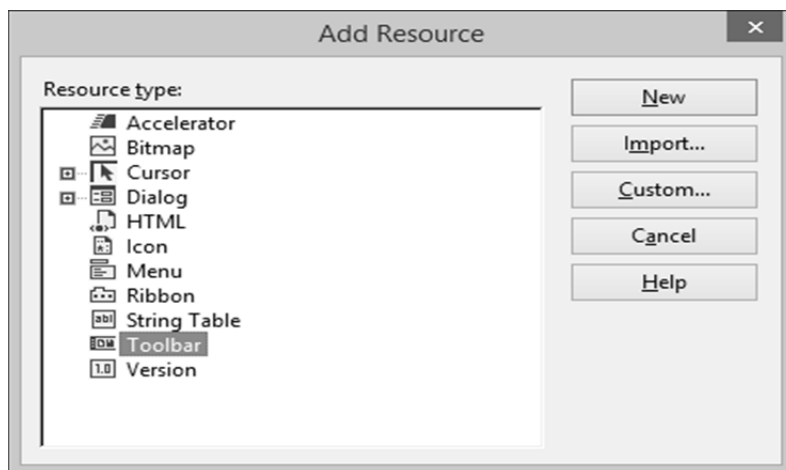
---

A **toolbar** is a Windows control that allows the user to perform some actions on a form by clicking a button instead of using a menu. It has the following characteristics:

- A toolbar provides a convenient group of buttons that simplifies the user's job by bringing the most accessible actions as buttons.
- A toolbar can bring such common actions closer to the user.
- Toolbars usually display under the main menu.
- They can be equipped with buttons but sometimes their buttons or some of their buttons have a caption.
- Toolbars can also be equipped with other types of controls.

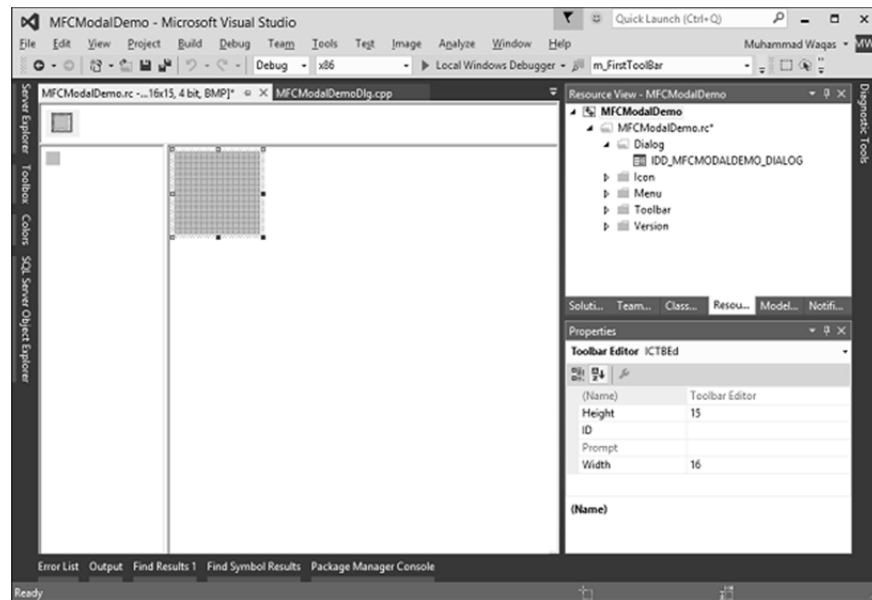
To create a toolbar, follow the steps given below.

**Step 1**– Right-click on your project and select Add → Resources. Which gives the Add Resources dialog box (Fig.7.23) as given below.



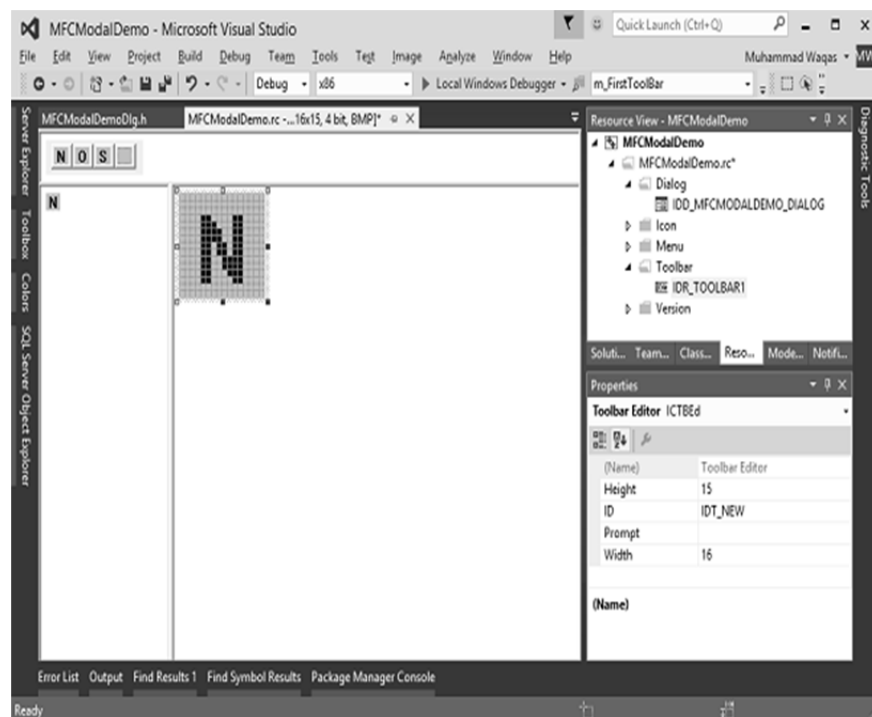
**Fig. 7.23** Add resource dialog box

**Step 2**– Select Toolbar and click New and it will show the following screen (Fig. 7.24).



**Fig. 7.24 Add resource toolbar**

**Step 3**– Design your toolbar in the designer as shown in the following screenshot and specify the IDs as well.



**Fig. 7.25 Designing the toolbar in the design**



## **CHECK YOUR PROGRESS**

- What do you understand by an accelerator?
- Write the main benefits of menus.
- Discuss about the significance of toolbars.

---

## **7.11 SUMMARY**

---

The *Microsoft Foundation Class (MFC)* library provides a set of functions, constants, data types, and classes to simplify creating applications for the Microsoft Windows operating systems. The *Document/View architecture* is the foundation used to create applications based on the MFCs' library. It allows you to make distinct the different parts that compose a computer program including what the user sees as part of the application and the document a user would work on. This is done through a combination of separate classes that work as an ensemble.

The *Single Document Interface* or SDI refers to a document that can present only one view to the user. This means that the application cannot display more than one document at a time. If you want to view another type of document of the current application, you must create another instance of the application. Notepad and WordPad are good examples of SDI applications. An application is referred to as a *Multiple Document Interface*, or MDI, if the user can open more than one document in the application without closing it. To provide this functionality, the application provides a parent frame that acts as the main frame of the computer program. Inside this frame, the application allows creating views with individual frames, making each view distinct from the other.

*Serialization* is the process of writing or reading an object to or from a persistent storage medium such as a disk file. Serialization is ideal for situations where it is desired to maintain the state of structured data (such as C++ classes or structures) during or after the execution of a program.

Separating from the view is intended with the interactions between documents and view. It is really routing that takes place between the document object and view object. Basically the data of the user is maintained by the document and the way it is presented to the user by means of view. The *CFormView* class is used as the base class for the users' views.

A *resource* is a text file that allows the compiler to manage objects such as pictures, sounds, mouse cursors, dialog boxes, etc. Microsoft Visual Studio makes creating a resource file particularly easy by providing the necessary tools in the same environment used to program. This means, you usually do not have to use an external application to create or configure a resource file.

The MFC Application Wizard generates an application that, when compiled, implements the basic features of a Windows executable (.exe) application. The MFC starter application includes C++ source (.cpp) files,

resource (.rc) files, header (.h) files, and a project (.vcxproj) file. The code that is generated in these starter files is based on MFC. Depending on the options selected, the wizard creates additional files in the project.

An *access key* is a letter that allows the user to perform a menu action faster by using the keyboard instead of the mouse. This is usually faster because the user would not need to position the mouse anywhere, which reduces the time it takes to perform the action. An *Accelerator Table* is a list of items where each item of the table combines an identifier, a shortcut key, and a constant number that specifies the kind of accelerator key. Just like the other resources, an accelerator table can be created manually in a .rc file.

*Menus* allow to arrange commands in a logical and easy-to-find fashion. With the Menu editor, menus can be created and edited by working directly with a menu bar that closely resembles the one in the finished application. A *toolbar* is a Windows control that allows the user to perform some actions on a form by clicking a button instead of using a menu.

---

## 7.12 TERMINAL QUESTIONS

---

1. What do you understand by Microsoft foundation classes? Explain.
2. Compare SDI and MDI.
3. Write a short note on serialization.
4. Write a short note on visual C++ resources.
5. Discuss MFC application wizard.
6. What do you understand by accelerators? Explain briefly.
7. Explain shortcut keys.
8. What do you understand by a menu? Write the steps to create a menu.
9. Write a short note on the use of toolbar.

---

# UNIT-8 GRAPHICS AND MULTIMEDIA

---

## Structure

- 8.0 Introduction
- 8.1 Objectives
- 8.2 Working with Graphics
- 8.3 Consoles
- 8.4 Multitasking Process and Threads
- 8.5 Drawing Graphics in Windows
- 8.6 Clipboards
- 8.7 Printing Graphics and Text
- 8.8 Creating Animations with Picture Clip Control
- 8.9 Summary
- 8.10 Terminal Questions

---

## 8.0 INTRODUCTION

---

Programming graphics is a lot harder than most people think. It involves tons of knowledge. More precisely, you need to know the concepts behind data structures, math, and be fairly good at some programming language. If you look at your computer screen (look closer), you'll notice that it is made up of little colored dots. These dots are called pixels. Each pixel has a specific color, and location on screen. Now, a simple definition of graphics programming is to make sure that right pixels appear in right places at right times. If one can make this happen, one can consider himself/herself graphics programmers.

The above definition is easier said than done. Keeping track of each pixel is next to impossible without some well-designed algorithms. And that's what most of this document is all about, algorithms! Code examples will mostly be in Java, since it's the only system independent language capable of graphics in comparison to the things in C/C++.

Computer screen is two dimensional; each pixel on the screen has some location, illustrated by the  $x$ , and  $y$  values. Where  $x$  is the horizontal offset from the left side of the screen, and  $y$  is the vertical offset from the top (or sometimes the bottom), of the screen. So, a location of  $x = 0$ , and  $y = 0$ , is the top left corner of the computer screen. Knowing about the  $x$ , and  $y$ , screen can be considered as a two dimensional array. With  $x$ , horizontal locations and  $y$  vertical locations. Each value inside the array represents a pixel. To plot a pixel on the screen at say  $x = 75$  and  $y = 100$ , pixel information can be put into the two dimensional array, at location  $[100][75]$ .

Now, to see the more abstract way, let us think of computer screen as a one dimensional array, kind of like starting at the upper left corner going right till end of line, and starting again on the other line, and continuing like that until it hits the lower right corner. To use this approach to plot pixels, we'll also need to know the width of the screen. Luckily, most of the time we know the width and height of the screen exactly. For example, lets name our one dimensional array "screen", so, to plot a pixel of color "color" at location  $x = 75$ , and  $y = 100$ , we'd do something like this: `screen[y * width + x] = color`. As you can see, it is exactly the same thing as `screen[y][x] = color`, only we're specifying the width manually.

Most of the times, working with graphics means working with a pointer. A pointer can be to memory, or directly to a video device. Our functions that "do graphics" don't have to be concerned with where the pointer points, we just need to draw pixels into that pointer (or it's offsets) as though it was a real screen.

---

## 8.1 OBJECTIVES

---

At the end of this unit you will come to know about the following

- Working with Graphics
- Consoles
- Multitasking process and threads
- Drawing Graphics in Windows
- Filling figures with colors and patterns
- Clipboard drag and drops
- Using clipboards to transfer images between applications
- Printing graphics and text
- Creating animations with Picture clip control

---

## 8.2 WORKING WITH GRAPHICS

---

A graphic is an image or visual representation of an object. Therefore, computer graphics are simply images displayed on a computer screen. Graphics are often contrasted with text, which is comprised of characters, such as numbers and letters, rather than images. Computer graphics can be either two or three-dimensional. Early computers only supported 2D monochrome graphics, meaning they were black and white (or black and green, depending on the monitor). Eventually, computers began to support color images. While the first machines only supported 16 or 256 colors, most computers can now display graphics in millions of colors.

2D Graphics come in two flavors—raster and vector. Raster graphics are the most common and are used for digital photos, Web graphics, icons, and other

types of images. They are composed of a simple grid of pixels, which can each be a different color. Vector graphics, on the other hand are made up of paths, which may be lines, shapes, letters, or other scalable objects. They are often used for creating logos, signs, and other types of drawings. Unlike raster graphics, vector graphics can be scaled to a larger size without losing quality.

3D Graphics started to become popular in the 1990s, along with 3D rendering software such as CAD and 3D animation programs. By the year 2000, many video games had begun incorporating 3D graphics, since computers had enough processing power to support them. Now most computers come with a 3D video card that handles all the 3D processing. This allows even basic home systems to support advanced 3D games and applications.

The graphics library used in Turbo C++ was called BGI (Borland Graphics Interface). There are a few "clones" of the library for Visual C++, MinGW, and other compilers, the most popular called WinBGI. Just for the record, the BGI is very old and it would only be recommended for compatibility reasons; for example, if you have some old Turbo C++ code you'd like to run in a modern OS, or if you're very proficient with BGI and don't have time to learn a new library. For developing a new application from scratch, I'd recommend using one of the various modern multiplatform graphics libraries that are being constantly updated. Moreover, OpenCV does anything the BGI does, and much more, much better. The basic drawing functions (lines, circles, etc) are just as easy to use as the ones in BGI, but it also has a ton of image processing functions that BGI lacks.

The Graphics Device Interface (GDI) is a Microsoft Windows application programming interface and core operating system component responsible for representing graphical objects and transmitting them to output devices such as monitors and printers. GDI is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes. It is not directly responsible for drawing windows, menus, etc.; that task is reserved for the user subsystem, which resides in user32.dll and is built atop GDI. Other systems have components that are similar to GDI, for example macOS' Quartz and X Window System's Xlib/XCB.

Before you can draw lines and shapes, render text, or display and manipulate images with GDI+, you need to create a Graphics object. The Graphics object represents a GDI+ drawing surface, and is the object that is used to create graphical images.

There are two steps in working with graphics

1. Creating a Graphics object.
2. Using the Graphics object to draw lines and shapes, render text, or display and manipulate images.

**Note:-** Windows GDI+ is a class-based API for C/C++ programmers. It enables applications to use graphics and formatted text on both the video display and the printer. Applications based on the Microsoft Win32 API do not access graphics hardware directly. Instead, GDI+ interacts with device drivers on behalf of

applications.

---

## 8.2.1 CREATING A GRAPHICS OBJECT

---

A graphics object can be created in a variety of ways

- Receive a reference to a graphics object as part of the `PaintEventArgs` in the `Paint` event of a form or control. This is usually how you obtain a reference to a graphics object when creating painting code for a control. Similarly, you can also obtain a graphics object as a property of the `PrintPageEventArgs` when handling the `PrintPage` event for a `PrintDocument`.

-or-

- Call the `CreateGraphics` method of a control or form to obtain a reference to a `Graphics` object that represents the drawing surface of that control or form. Use this method if you want to draw on a form or control that already exists.

-or-

- Create a `Graphics` object from any object that inherits from `Image`. This approach is useful when you want to alter an already existing image.

---

## 8.2.2 USING THE GRAPHICS OBJECT

---

After it is created, a `Graphics` object may be used to draw lines and shapes, render text, or display and manipulate images. The principal objects that are used with the `Graphics` object are:

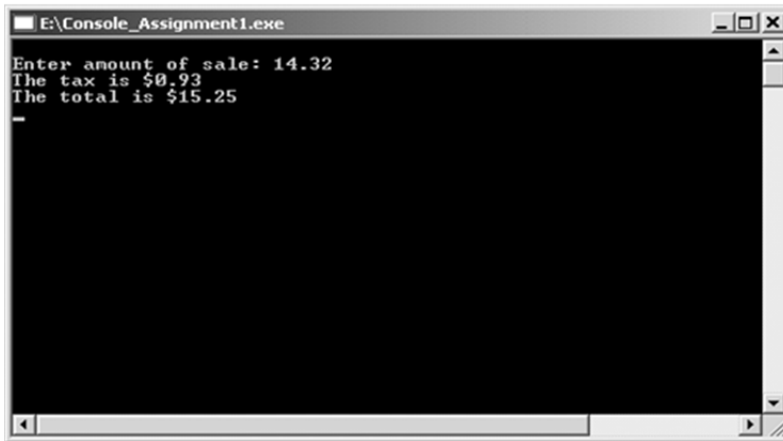
- **The Pen class**— Used for drawing lines, outlining shapes, or rendering other geometric representations.
- **The Brush class**— Used for filling areas of graphics, such as filled shapes, images, or text.
- **The Font class**— Provides a description of what shapes to use when rendering text.
- **The Color structure** — Represents the different colors to display.

---

## 8.3 CONSOLES

---

A console application is a program which runs in a command prompt window. An example of a console application is shown in Fig 8.1. Console programs do not have the flash, nor the event-driven capabilities of a Windows application, however, they still have their place. For example, where screen space is limited such as a bank ATM, or a device with a very simple display type.



**Fig. 8.1 Console application**

Both input and output statements in console applications reside in the same class, the `System.Console` class. The class method mostly used to perform console output is the `WriteLine` method. The generic format for the `WriteLine` method is:

```
Console.WriteLine ("string to write to console")
```

As we have seen in our error to write the string "Hello World!" to the console, we use the statement:

```
Console.WriteLine ("Hello World!")
```

It should be noted that the `WriteLine` method also writes a carriage return/line feed (CRLF) following the string argument. The similar `Console.Write` method behaves as does the `WriteLine` method except without the CRLF.

The class method most frequently used to perform console input is the `ReadLine` method. The `ReadLine` method reads the next line of characters from the input stream and assigns these to the variable via the assignment operator. The `Console.ReadLine` method returns a `String` data type. When using the `ReadLine` method, the program will pause and wait until the Enter key is pressed before continuing (i.e. waits for data to be input). The generic format for the `ReadLine` method is:

```
string_variable = Console.ReadLine()
```

Data input can be combined with type conversion to form a single line statement (although there are still 2 parts to the single line statement). An example looks as follows:

```
decInput = Convert.ToDecimal (Console.ReadLine())
```

The `Console.ReadLine` statement performs the data input, then passes the input `String` data directly to the `Convert` method, which performs the conversion and assigns the converted data to the `Decimal` variable. Notice there is no intermediate `String` variable for the `ReadLine`, the data flows directly from the `ReadLine` into the `Convert` method. The `Console` class belongs to the `System` namespace. Since the `System` namespace is the top most hierarchical namespace, this namespace is imported by default so the `System` can be omitted.

---

## **8.4 MULTITASKING PROCESS AND THREADS**

---

Multitasking is the ability of an operating system (OS) to run various programs simultaneously. But at the background OS uses a round-robin approach i.e. it provides “time slices” for each running process; and a user thinks that all the programs are running concurrently. Multitasking is a very old concept which was even present during the mainframe computers too. Then in those mainframes, jobs were submitted and executed one-by-one accordingly. It took some time to become a reality or enter into personal computers. First of all, 16-bit versions of the OS supported multitasking. On the other hand, 32-bit OS supported multitasking as well as a new concept – multithreading. Multithreading is the ability of a program to multitask within itself. The program can be divided further into smaller separate parts/pieces of execution called threads which also seem to run concurrently.

---

### **8.4.1 MULTITASKING UNDER DOS**

---

Disk Operating System (DOS) does not support the multitasking rather it is used to do the same work by terminate-and-stay-resident (TSR) programs. Some TSRs, like print spoolers, hooked into the hardware timer interrupt to perform true background processing. Some software vendors attempted to mold task-switching or multitasking shells on top of DOS. Windows 3.x and older were mostly 16-bit (with the exception of Win32s), were more dependent on DOS, and used only cooperative multitasking - that's the one where they don't force a running program to switch out; they wait for the running program to yield control (basically, say "I'm done" by telling the OS to run the next program that's waiting). Windows 3.1 uses cooperative multitasking - meaning that each application that is in the process of running is instructed to periodically check a message queue to find out if any other application is asking for use of the CPU and, if so, to yield control to that application. However, many Windows 3.1 applications would check the message queue only infrequently, or not at all, and monopolize control of the CPU for as much time as they required. A preemptive multitasking system like Windows 95 will take CPU control away from a running application and distribute it to those that have a higher priority based on the system's needs.

---

### **8.4.2 NON-PREEMPTING MULTITASKING**

---

In general rule of Non-Preempting Scheduling, a program would sit idle in memory until it receives a message; which were often direct or indirect result of user input through the keyboard or mouse and after processing the message, program returned the control to Windows. All the limitations of the DOS were overcome in Windows 1.0 introduced in 1985. The main thing that Windows performed, was to move blocks within the physical memory – a priority for multitasking. It is also possible in Windowing environment to move quickly data from one program to another. Data transfer has been supported in Windows in different ways. Non-Preempting Multitasking was supported by 16-bit OS. This type of multitasking was possible only due the message-driven architecture of



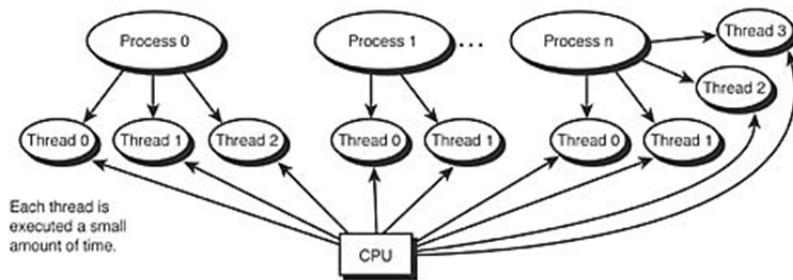
Windows. Windows used Preempting scheduling for DOS programs. This is also known as cooperative multitasking.

---

### 8.4.3 MULTITHREADING

---

In a multithreaded environment, programs can split themselves into separate pieces (called threads of execution) that run concurrently (Fig 8.2). This was the best solution for the problem of queuing or serialization in the presentation manager. In terms of code, a thread is simply represented by a function which might also call other functions in the program. Generally, a program starts execution with its main (or primary) thread, which is main, in traditional C/C++ programming. This main of C/C++ is WinMain in Windows programming. Once the main thread is created, it may create other threads too by making a system call specifying the name of the initial function. The OS preemptively switches control among the threads in same way it does among processes.



**Fig. 8.2 Process and thread**

(Source: [http://www.yaldex.com/games-programming/0672323699\\_ch02levlsec2.html](http://www.yaldex.com/games-programming/0672323699_ch02levlsec2.html))

The limitations of Presentation Manager or PM, provided programmers with essential clues to understanding how to use multiple threads of execution in a program running under a graphical environment. So here's the architecture of your programs:

- the primary thread creates all the windows that a program needs,
- includes all the window procedures for these windows,
- and processes all the messages for these windows.

Any other threads are simply background problems. They do not interact with the user except through communication with the primary thread. One way to think of this is that the primary thread handles user input (and other messages), perhaps creating secondary threads in the process. These additional threads do the non-user-related tasks.

In other words, program's primary thread is a governor, and secondary threads are the governor's staff. The governor delegates all the big jobs to his or her staff while maintaining contact with the outside world. Because they are staff

members, the secondary threads do not hold their own press conferences. They silently do their work, report back to the governor, and await their next assignment. Threads within a particular program are all parts of the same process, so they share the process's resources, such as memory and open files. Because threads share the program's memory, they also share static variables. However, each thread has its own stack, so automatic variables are unique to each thread. Each thread also has its own processor state (and math coprocessor state) that is saved and restored during thread switches.

## 8.4.4 THE EVENT MODEL

Windows is a multitasking/multithreaded OS, but it is also an event-driven. Unlike DOS programs, most Windows programs sit and wait for the user to do something, which fires an event, and then Windows responds to the event and takes action which can be seen graphically in Fig 8.3. It depicts a number of application windows, each sending their events or messages to Windows to be processed. Windows does some of the processing, but most of the messages or events are passed through to your application program for processing.

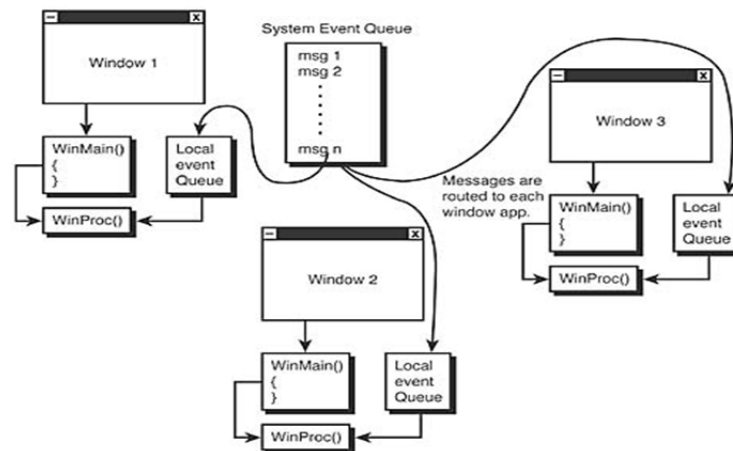


Fig 8.3 Event model

(Source: [http://www.yaldex.com/games-programming/0672323699\\_ch02lev1sec2.html](http://www.yaldex.com/games-programming/0672323699_ch02lev1sec2.html))

### CHECK YOUR PROGRESS

- What do you understand by Computer Graphics?
- Briefly define the term GDI.
- What is console application?
- Give the meaning of multitasking.
- Compare process and thread.

---

## 8.5 DRAWING GRAPHICS IN WINDOWS

---

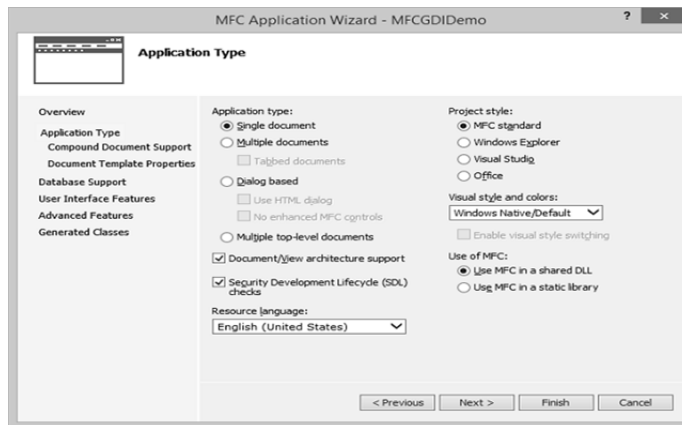
Windows provides a variety of drawing tools to use in device contexts. It provides pens to draw lines, brushes to fill interiors, and fonts to draw text. MFC provides graphic-object classes equivalent to the drawing tools in Windows. A device context is a Windows data structure containing information about the drawing attributes of a device such as a display or a printer. All drawing calls are made through a device-context object, which encapsulates the Windows APIs for drawing lines, shapes, and text. Device contexts allow device-independent drawing in Windows. Device contexts can be used to draw to the screen, to the printer, or to a metafile. CDC is the most fundamental class to draw in MFC. The CDC object provides member functions to perform the basic drawing steps, as well as members for working with a display context associated with the client area of a window.

---

### 8.5.1 LINES

---

Step 1– Consider a simple example to draw a line by creating a new MFC based single document project with MFCGDIDemo name (Fig. 8.4).



**Fig. 8.4 MFC Application wizard**

Step 2– Once the project is created, go the Solution Explorer and double click on the MFCGDIDemoView.cpp file under the Source Files folder.

Step 3– Draw the line as shown below in CMFCGDIDemoView::OnDraw() method.

```
void CMFCGDIDemoView::OnDraw(CDC* pDC) {  
    pDC->MoveTo(95, 125);  
    pDC->LineTo(230, 125);  
    CMFCGDIDemoDoc* pDoc = GetDocument();  
    ASSERT_VALID(pDoc);  
    if (!pDoc)
```

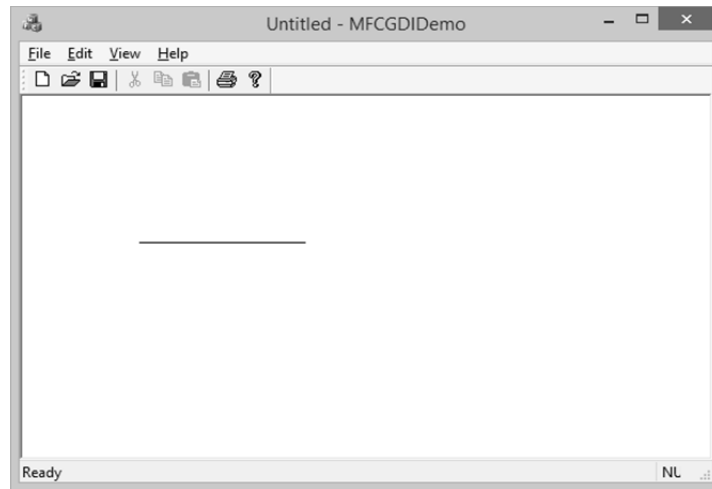
```

return;

// TODO: add draw code for native data here
}

```

Step 4– Run this application and see the following output shown in the Fig. 8.4.



**Fig. 8.4 Output of Line program**

Step 5– The `CDC::MoveTo()` method is used to set the starting position of a line.

When using `LineTo()`, the program starts from the `MoveTo()` point to the `LineTo()` end. After `LineTo()` when you do not call `MoveTo()`, and call again `LineTo()` with other point value, the program will draw a line from the previous `LineTo()` to the new `LineTo()` point.

Step 6 – To draw different lines, you can use this property as shown in the following code.

```

void CMFCGDI DemoView::OnDraw(CDC* pDC) {
    pDC->MoveTo(95, 125);
    pDC->LineTo(230, 125);
    pDC->LineTo(230, 225);
    pDC->LineTo(95, 325);

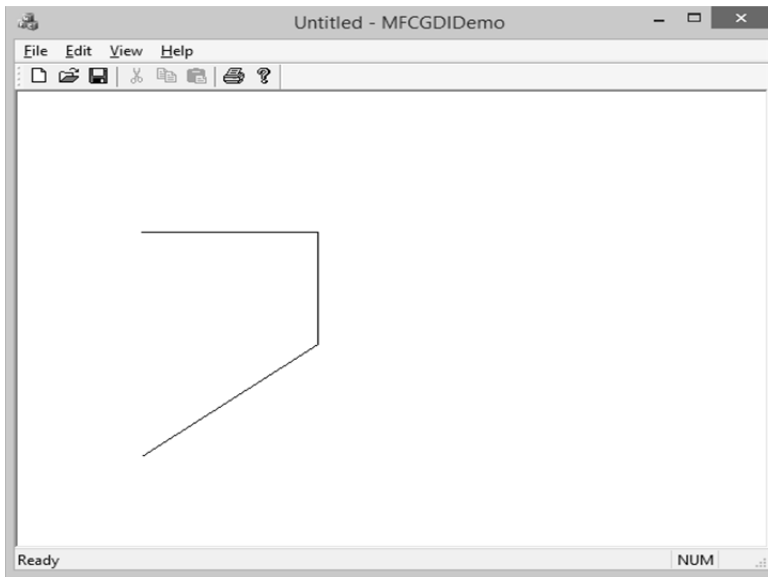
    CMFCGDI DemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}

```

Step 7 – Run this application and see the output shown in the Fig.8.5.



**Fig. 8.5 Output of Line program**

---

## 8.5.2 POLYLINES

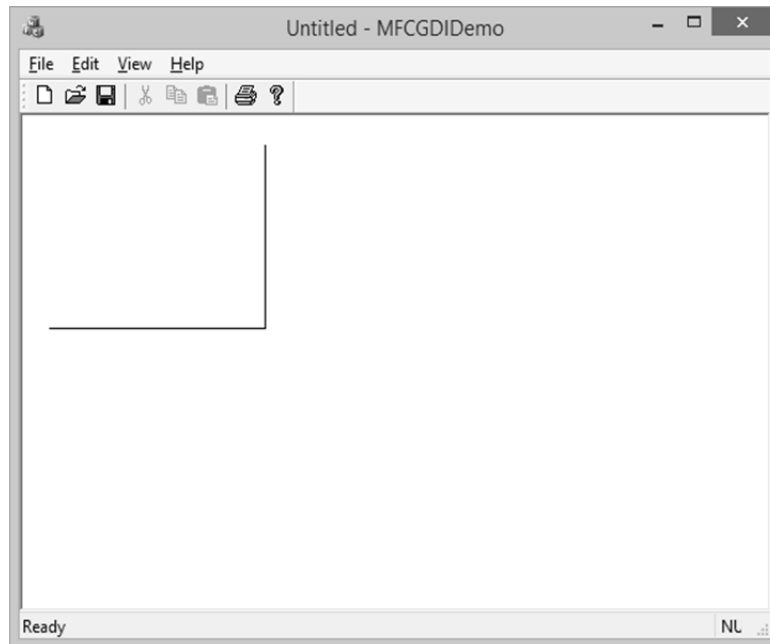
---

A polyline is a series of connected lines. The lines are stored in an array of POINT or CPoint values. To draw a polyline, use the CDC::Polyline() method. To draw a polyline, at least two points are required. If you define more than two points, each line after the first would be drawn from the previous point to the next point until all points have been included.

Step 1– Let us look into a simple example.

```
void CMFCGDIDemoView::OnDraw(CDC* pDC) {
    CPoint Pt[7];
    Pt[0] = CPoint(20, 150);
    Pt[1] = CPoint(180, 150);
    Pt[2] = CPoint(180, 20);
    pDC->Polyline(Pt, 3);
    CMFCGDIDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: add draw code for native data here
}
```

Step 2– When you run this application, you will see the following output (Fig. 8.6).



**Fig. 8.6 Output of PolyLines program**

---

### 8.5.3 RECTANGLES

---

A rectangle is a geometric figure made of four sides that compose four right angles. Like the line, to draw a rectangle, you must define where it starts and where it ends. To draw a rectangle, you can use the `CDC::Rectangle()` method.

Step 1– Let us look into a simple example.

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

Step 2– When you run this application, you will see the following output (Fig. 8.7).



**Fig. 8.7 Output of Rectangles program**

---

## 8.5.4 SQUARES

---

A square is a geometric figure made of four sides that compose four right angles, but each side must be equal in length.

Let us look into a simple example.

```
void CMFCGDI DemoView::OnDraw(CDC* pDC) {
    pDC->Rectangle(15, 15, 250, 250);

    CMFCGDI DemoDoc* 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 (Fig. 8.8).



**Fig. 8.8 Output of Square program**

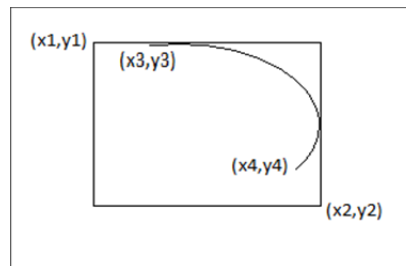
---

## 8.5.5 ARCS

---

An arc is a portion or segment of an ellipse, meaning an arc is a non-complete ellipse. To draw an arc, you can use the `CDC::Arc()` method (Fig. 8.9).

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



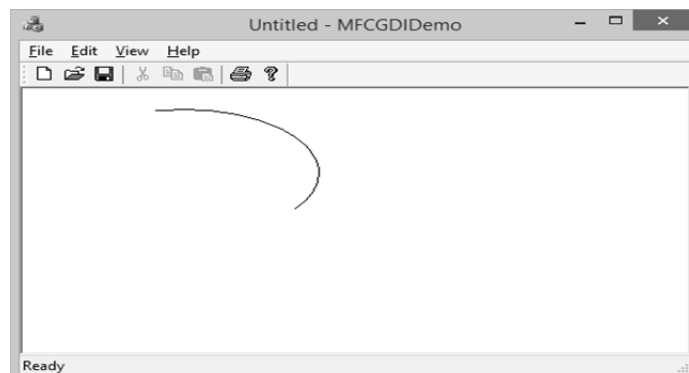
**Fig. 8.9 Arcs' coordinates**

The `CDC` class is equipped with the `SetArcDirection()` method. Here is the syntax – `int SetArcDirection(int nArcDirection)`

Step 1 – Let us look into a simple example.

```
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  
}
```

Step 2– When you run this application, you will see the following output (Fig. 8.10).



**Fig. 8.10 Output of Arc program**



---

## 8.5.6 SETTING COLORS

---

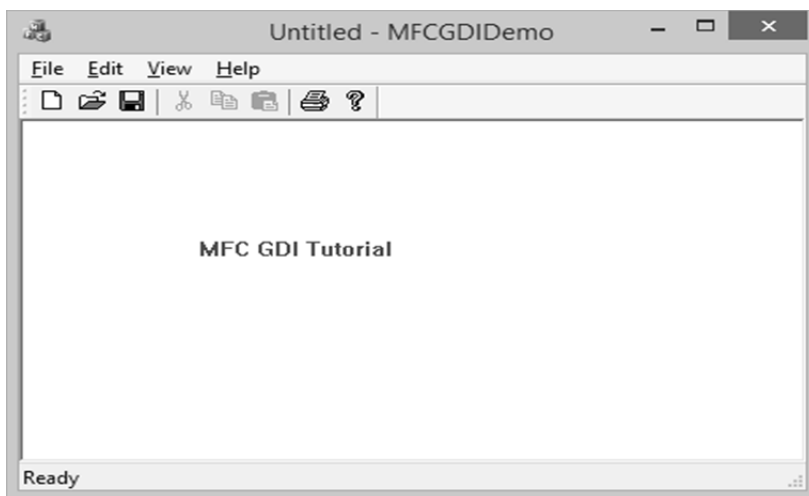
The color is one the most fundamental objects that enhances the aesthetic appearance of an object. The color is a non-spatial object that is added to an object to modify some of its visual aspects. The MFC library, combined with the Win32 API, provides various actions you can use to take advantage of the various aspects of colors. The RGB macro behaves like a function and allows you to pass three numeric values separated by a comma. Each value must be between 0 and 255 as shown in the following code:

```
void CMFCGDI DemoView::OnDraw(CDC* pDC) {  
    COLORREF color = RGB(239, 15, 225);  
}
```

Let us look into a simple example.

```
void CMFCGDI DemoView::OnDraw(CDC* pDC) {  
    COLORREF color = RGB(239, 15, 225);  
    pDC->SetTextColor(color);  
    pDC->TextOut(100, 80, L"MFC GDI Tutorial", 16);  
  
    CMFCGDI DemoDoc* 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.



**Fig. 8.11** Output of Setting colors

---

## 8.5.7 FONTS

---

CFont encapsulates a Windows graphics device interface (GDI) font and provides member functions for manipulating the font. To use a CFont object, construct a CFont object and attach a Windows font to it, and then use the object's member functions to manipulate the font. Here is a list of methods in CFont class (table 8.1).

**Table 8.1 Methods in CFont Class**

S.No	Name	Description
1	CreateFont	Initializes a CFont with the specified characteristics.
2	CreateFontIndirect	Initializes a CFont object with the characteristics given in a LOGFONT structure.
3	CreatePointFont	Initializes a CFont with the specified height, measured in tenths of a point, and typeface.
4	CreatePointFontIndirect	Same as CreateFontIndirect except that the font height is measured in tenths of a point rather than logical units.
5	FromHandle	Returns a pointer to a CFont object when given a Windows HFONT.
6	GetLogFont	Fills a LOGFONT with information about the logical font attached to the CFont object.

---

## 8.5.8 PENS

---

A pen is a tool used to draw lines and curves on a device context. In the Graphics programming, a pen is also used to draw the borders of a geometric closed shape such as a rectangle or a polygon. Microsoft Windows considers two types of pens — cosmetic and geometric. A pen is referred to as cosmetic when it can be used to draw only simple lines of a fixed width, less than or equal to 1 pixel. A pen is geometric when it can assume different widths and various ends. MFC provides a class CPen which encapsulates a Windows graphics device interface (GDI) pen.

---

## 8.5.9 BRUSHES

---

A brush is a drawing tool used to fill out closed shaped or the interior of lines. A brush behaves like picking up a bucket of paint and pouring it somewhere. MFC provides a class CBrush which encapsulates a Windows graphics device interface (GDI) brush. List of methods in CBrush class has been shown in Table 8.2.

**Table 8.2 List of methods for brush class**

S.No	Name	Description
1	CreateBrushIndirect	Initializes a brush with the style, color, and pattern specified in a LOGBRUSH structure.
2	CreateDIBPatternBrush	Initializes a brush with a pattern specified by a device-independent bitmap (DIB)
3	CreateHatchBrush	Initializes a brush with the specified hatched pattern and color.
4	CreatePatternBrush	Initializes a brush with a pattern specified by a bitmap.
5	CreateSolidBrush	Initializes a brush with the specified solid color.
6	CreateSysColorBrush	Creates a brush that is the default system color
7	FromHandle	Returns a pointer to a CBrush object when given a handle to a Windows HBRUSH object
8	GetLogBrush	Gets a LOGBRUSH structure

Example:

```
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 ;
}
```

**Output(Fig.8.12):**



**Fig. 8.12 Output of Brush example**

---

## **8.6 CLIPBOARDS**

---

The clipboard is a buffer that some operating systems provide for short-term storage and transfer within and between application programs. The clipboard is usually temporary and unnamed, and its contents reside in the computer's RAM. The clipboard is sometimes called the paste buffer. In Windows, but also in other operating systems like Android or Mac OS X, the clipboard is a special location in the PC or device memory, that is used as a temporary storage area for any information that is copied. Once some data is copied into the clipboard, it can be pasted somewhere else, in the same application from which it was copied, or in a different one, as long as it knows how to work with that type of data. It is also referred to as pasteboard, is a special location in computer's memory that temporarily stores data that was cut or copied from a document. This data can then be pasted to a new location. The clipboard will hold its information until you cut or copy something else, or log out of the computer. For example, a user may copy information from a word processor and paste that information into an e-mail message.

Many operating systems include "clipboard viewers" that display what information is currently being stored in the clipboard. These utilities may also be used to configure the clipboard with permissions, or view the clipboard's history. Unfortunately, Microsoft has decided not to include any clipboard viewer in Microsoft Windows Vista, 7, 8, and 10. To view the contents of the clipboard, you need to download a third-party utility or app.

Microsoft Windows 2000 and XP users may find it difficult to locate the clipboard because it was renamed to the Clipbook viewer. It can be located by opening Windows Explorer, opening the "Winnt" or "Windows" folder, then the "System32" folder. Find and double click the clipbrd.exe file. Users can also click Start, Run, type clipbrd and press Enter to execute this program. Microsoft Windows 95, 98, NT 4.0, and ME come installed with a clipboard viewer that can be run by clicking Start, Programs, System Tools, and clicking Clipboard Viewer. The clipboard viewer is also executable through the clipbrd.exe file in the Windows directory.

An operating system that supports a clipboard provides an API by which

programs can specify cut, copy and paste operations. It is left to the program to define methods for the user to command these operations, which may include keybindings and menu selections. When an element is copied or cut, the clipboard must store enough information to enable a sensible result no matter where the element is pasted. Application programs may extend the clipboard functions that the operating system provides. A clipboard manager may give the user additional control over the clipboard. Specific clipboard semantics vary among operating systems, can also vary between versions of the same system, and can sometimes be changed by programs and by user preferences.

A clipboard manager is a computer program that adds functionality to an operating system's clipboard. Many clipboards provide only one buffer for the "copy and paste" function, and it is overwritten by each new "copy" operation. The main task of a clipboard manager is to store data copied to the clipboard in a way that permits extended use of the data. Clipboard managers enhance the basic functions of cut, copy, and paste operations with one or more of the following features:

- Multiple buffers and the ability to merge, split, and edit their contents
- Selecting which buffer "cut" or "copy" operations should store data in
- Selecting which buffer(s) "paste" operations should take data from
- Handling formatted text, tabular data, data objects, media content, and URLs
- Saving copied data to long term storage
- Indexing or tagging of clipped data
- Searching of saved data

---

## **8.6.1 THE STANDARD CLIPBOARD DATA FORMATS**

---

A window can place more than one object on the clipboard, each representing the same information in a different clipboard format. Users need not be aware of the clipboard formats used for an object on the clipboard. The clipboard formats defined by the system are called standard clipboard formats. These clipboard formats are described in Standard Clipboard Formats. Many applications work with data that cannot be translated into a standard clipboard format without loss of information. These applications can create their own clipboard formats. A clipboard format that is defined by an application, is called a registered clipboard format. For example, if a word-processing application copied formatted text to the clipboard using a standard text format, the formatting information would be lost. The solution would be to register a new clipboard format, such as Rich Text Format (RTF).

---

### **8.6.1.1 FORMATS**

---

You can usually copy/paste within a program, and between any two programs where a common "format" can be agreed upon. For example, you can copy text from a web browser and paste into a word processor or text editor. But

you will not be able to copy an image from a paint program and paste into a simple text editor like Notepad. That's because they don't have any way to convert the image into plain text. Here are the most common data formats:

- *Text*– plain text, with no font, formatting, color, images. When pasted into a word processor, will appear just as if you'd typed it from the keyboard.
- *Rich Text Format*– Text with formatting, such as you'd use when copying within a word processor. Contains fonts/formatting, which may clash with the formatting already present within the word processor.
- *HTML*– Text and images copied from browsers, also contains formatting and tables. Can seriously contaminate formatting and style within a word-processing document.
- *Bitmap*– Almost all image data on the clipboard uses this format, even if you copy a Jpeg, GIF, or PNG from a browser window. Bitmap is the “rosetta stone” of the clipboard, as far as images go. Screen grabs are also sent to the clipboard as Bitmap.
- *Picture*– The “other” image format, used for vector drawings like you make with a CAD program, or the drawing tool in Microsoft Word.
- *File Pointers*– These are on the clipboard when you “copy” files or folders within Windows Explorer. This lets you “paste” into another folder, which causes Windows to copy or move the files. Note that the files themselves aren't really on the clipboard – it's just a “pointer” that describes where the files are. It would take far too long, to actually place the contents of the files onto the clipboard.

Moreover, Windows supports various formats which have identifiers in windows header files. The most common of these are as given below:

1. *CF\_TEXT*– A simplest form of clipboard data, is a null-terminated ANSI character string containing a carriage return and a linefeed character at the end of each line.
2. *CF\_BITMAP*– It is a device dependent bitmap which is transferred to the clipboard using the bitmap handle.
3. *CF\_METAFILEPICT*– It is a metafile with some extra information in the form of a small structure of type *METAFILEPICT*. The program transfers a metafile picture to the clipboard using the handle to a memory block containing this particular structure.
4. *CF\_SYLK*– It is a memory block containing data in the Data Interchange Format. It is used to exchange the data among Microsoft Multiplan, Chart and Excel programs.

---

## 8.6.2 TRANSFERRING TEXT TO THE CLIPBOARD

---

Let's assume that you want to transfer a character string to the clipboard and that you have a pointer (Called pString) to this string. You want to transfer iLength bytes of this string. First, allocate a moveable memory block of iLength

size by using GlobalAlloc. Include room for a terminating NULL:

```
hGlobalMemory= Globalalloc(GHND, iLength+1);
```

The value of bGlobalMemory will be NULL if the block could not be allocated. If the allocation is successful, lock the block to get a pointer to it:

```
pGlobalMemory=GlobalLock(hGlobalMemory);
```

Copy the character string into the memory block:

```
for (i=0;i<iLength;i++)
    *pGlobalMemory++=*pString++;
```

You don't need to add terminating NULL because the GHND flag for GlobalAlloc zeroes out the entire memory block during allocation. Unlock the block:

```
GlobalUnlock(hGlobalMemory)
```

Now you have a memory handle that references a memory block containing the NULL terminates text. To get this into the clipboard, open the clipboard and empty it:

```
OpenClipboard(hwnd);
EmptyClipboard();
```

Give the clipboard the memory handle using the CF\_TEXT identifier, and close the clipboard:

```
SetClipboardData(CF_TEXT, hGlobalMemory);
CloseClipboard();
```

---

## 8.6.3 OPENING AND CLOSING THE CLIPBOARD

---

Only one program can open the clipboard in one time. The purpose of the OpenClipboard is to prevent the clipboard contents from changing while a program is use the clipboard. OpenClipboard returns a Boolean value which indicates that whether the program is opened or not. If another program is open already it will return true and not allow to open the program. On the other hand, if it returns false it allows to open the program.

---

## 8.6.4 CLIPBOARD DRAG AND DROPS

---

Drag and drop operations in windows-based application can be enabled using or handling various events like DragEnter, DragLeave, and DragDrop. Working with the information available in the event arguments of these events, drag-and-drop operations can be easily used. User's cut/copy/paste support and user data transfer to the clipboard can also be implemented within the Windows-based applications using simple method calls.

---

### 8.6.4.1 DRAGGING DATA

---

All drag-and-drop operations begin with dragging. The functionality to enable data to be collected when dragging begins is implemented in the

DoDragDrop method. In most of the applications, the MouseDown event is used to start the drag operation because it is the most intuitive (most drag-and-drop actions begin with the mouse button being depressed).

**Note1:** - However, remember that any event could be used to initiate a drag-and-drop procedure.

**Note2:** - Certain controls have custom drag-specific events. The ListView and TreeView controls, for example, have an ItemDrag event.

In the MouseDown event for the control where the drag will begin, use the DoDragDrop method to set the data to be dragged and the allowed effect dragging will have. The control where the drag begins is a Button control, the data being dragged is the string representing the Text property of the Button control, and the allowed effects are either copying or moving. While a drag operation is in effect, you can handle the QueryContinueDrag event, which "asks permission" of the system to continue the drag operation. When handling this method, it is also the appropriate point for you to call methods that will have an effect on the drag operation, such as expanding a TreeNode in a TreeView control when the cursor hovers over it.

**Note:** - Any data can be used as a parameter in the DoDragDrop method like Text property of Button control.

---

#### **8.6.4.2 DROPPING DATA**

---

Once dragging data starts from a location on a Windows Form or control and want to drop it somewhere. The cursor changes when it crosses an area of a form or control that is correctly configured for dropping data. Any area within a Windows Form or control can be made to accept dropped data by setting the AllowDrop property and handling the DragEnter and DragDrop events.

For a drop operation steps required are:

- Set the AllowDrop property to true.
- In the DragEnter event for the control where the drop will occur, ensure that the data being dragged is of an acceptable type. The code then sets the effect that will happen when the drop occurs to a value in the DragDropEffects enumeration.

---

#### **8.6.5 USING CLIPBOARDS TO TRANSFER IMAGES BETWEEN APPLICATIONS**

---

The clipboard function is one of the most common features of Microsoft Windows. It's something you likely do a dozen—if not a hundred—times a day: Copy and Paste. Something's somewhere, and you want it someplace else, so you copy it then paste it in the new place—a modern version of Xeroxing a piece of paper, cutting out the text you wanted, and gluing (or pasting) it to the other document where you needed the text. Cutting or copying bits of text or an image from one application and then pasting it into another application is a process many of us perform on an almost daily basis. In many ways, it is fundamental to



our collaborative and social media-centric digital lives.

Windows 10 added a new feature where, if you tap the Win + Printscreen keys, a screenshot is captured and saved to a folder called Screenshots in the Picture library. This is by far the easiest way to capture a full screen screenshot of practically anything. The PrintScreen key, when tapped on its own, also captures your screen but the image goes to your clipboard. It's not saved as a file unless you paste it into an image editor. This is tedious so if you want a quicker way to save a clipboard image to a file on Windows 10, a free app called Paste InTo File will do the trick. The Copy Image command places a selections image into the clipboard and the internal buffer. The internal buffer may be pasted to the diagram, and the image (bitmap and metafile) in the clipboard may be pasted into other applications. To copy an image, select an object or region and execute Copy Image. This command is also available from the pop up menu (click the right mouse button on the diagram window to make the popup menu appear.)

The Cut Image command places images of a selection into the clipboard and the internal buffer, then deletes the selection from the diagram. Once the internal buffer is filled, the selection in it may be pasted into the diagram, and the clipboard image (bitmap and metafile) may be pasted into other applications. To cut an image, select an object or region and execute the Cut Image command. This command is also available from the pop up menu (click the right mouse button on the diagram window to make the popup menu appear. Paste Image causes a copy of the internal StateCAD buffer to be placed into the diagram. To paste, select a region into the buffer (via Cut Image or Copy Image), then select Paste Image. A green rectangle appears on the screen which should be moved to the region where the pasted image is desired. Click the left mouse button to complete the operation. Pasting may be canceled by typing [ESC] after executing the Paste Image command but before the click that finalizes the paste. Images may be pasted within StateCAD from the internal buffer. Images in the clipboard may not be pasted into StateCAD. Once StateCAD has been exited the internal buffer is flushed, but the clipboard retains whatever information it had up to that point. Pasting may be done between diagrams and within a diagram, as long as StateCAD is not exited. This command is also available from the pop up menu (click the right mouse button on the diagram window to make the popup menu appear.

When pasting named objects (states, logic, and vectors) new objects are assigned unique names. If the objects in the buffer are not present in the diagram, then the objects original names are used. Aliases are not renamed when pasted (or imported). Whenever a Cut Image or Copy Image command is executed on a selection, a full color image of the selection is copied to the clipboard (metafile and bitmap formats) and a description image is copied to the internal buffer. Once an image has been placed into the clipboard, it may be pasted into any Windows package which accepts bitmaps or metafiles from the clipboard. When an area is selected into the clipboard the zoom level is ignored. Anything sent to the clipboard is shown in the clipboard in its unzoomed size. This makes it easy to see information about the design (text etc.) when zoomed.

**Note:** - Images are copied as metafiles and bitmaps. Applications automatically select the best format.

When a selection is copied to the clipboard, control points and page breaks are suppressed. To copy the video screen to the clipboard, use the Windows feature, print screen ([CTRL]+[PRINT SCR]), or use print window ([ALT]+[PRINT SCR]) to copy the currently active window.

Nowadays Cloud Computing Technology has also become very popular as the images throughout the computers are available due to Cloud Computing. New cloud-powered clipboard experience included with the October 2018 Update.

---

## 8.6.6 LIMITATIONS OF CLIPBOARD

---

- The clipboard is simply a shared block of memory, and while it can contain the same chunk of data represented in a variety of formats, it can only hold one item at a time. So if you copy something that you want to paste, you need to paste it before you copy something else. Otherwise, your first item will be overwritten, and lost. This limitation can be overcome by the use of a Clipboard Extender, such as ClipMate.
- The limit of images for keeping in clipboard is 4MB.
- Once the file is copied and pasted somewhere, it can't be pasted to other desired location as it is not available on the device working upon.

**Note:** - It is impossible to completely "backup" the clipboard and restore it like it was, without impacting other programs, and causing a negative user experience.

### CHECK YOUR PROGRESS

- What do you understand by device context?
- Briefly define the standard clipboard data format.
- What are the limitations of clipboard?

---

## 8.7 PRINTING GRAPHICS AND TEXT

---

Printing from a Windows program usually involves some overhead, as well as some GDI calls to actually print something. So far, the only experience you've had with GDI is the processing of the WM\_PAINT message in the main event handler. Remember that GDI, or the Graphics Device Interface, is how all graphics are drawn under Windows when DirectX is not in use. Alas, you haven't yet learned how to actually draw anything on the screen with GDI, but this is very key because rendering on the screen is one of the most important parts of writing a video game. Basically, a game is just logic that drives a video display. In this section, the WM\_PAINT message has been revisited, cover some basic video concepts, and see how to draw text within window.

Understanding the WM\_PAINT message is very important for standard GDI graphics and Windows programming because most Windows programs' displays revolve around this single message. In a DirectX game this isn't true,

because DirectX, or more specifically DirectDraw or Direct3D, will do the drawing, but still need to know GDI to write Windows applications. The WM\_PAINT message is sent to window's WinProc() whenever the window's client area needs repainting. Until now, you haven't done much processing on this event. Here's the standard WM\_PAINT handler which have been used:

```
PAINTSTRUCT ps; // used in WM_PAINT
HDC hdc; // handle to a device context

case WM_PAINT:
{
    // simply validate the window
    hdc = BeginPaint(hwnd,&ps);
    // you would do all your painting here
    EndPaint(hwnd,&ps);
    // return success
    return(0);
} break;
```

When a window is moved, resized, or in some way graphically obscured by another window or event, some or all of the window's client area must be redrawn. When this happens, a WM\_PAINT message is sent and you must deal with it. In the case of the preceding code example, the calls to BeginPaint() and EndPaint() accomplish a couple of tasks. First, they validate the client area, and second, they fill the background of the window with the background brush defined in the Windows class that the window is originally created with. Now, if you want to do your own graphics within the BeginPaint()—EndPaint() call, you can. However, there is one problem: You will only have access to the portion of the window's client area that actually needs repainting. The coordinates of the invalid rectangle are stored in the rcPaint field of the ps (PAINTSTRUCT) returned by the call to BeginPaint():

```
typedef struct tagPAINTSTRUCT
{
    HDC hdc; // graphics device context
    BOOL fErase; // if TRUE then you must draw background
    RECT rcPaint; // the RECT containing invalid region
    BOOL fRestore; // internal
    BOOL fIncUpdate; // internal
    BYTE rgbReserved[32]; // internal
} PAINTSTRUCT;
```

---

## 8.8 CREATING ANIMATIONS WITH PICTURE CLIP CONTROL

---

An animation control is a window that displays an Audio-Video Interleaved (AVI) clip. An AVI clip is a series of bitmap frames like a movie. Animation controls can only display AVI clips that do not contain audio. One common use for an animation control is to indicate system activity during a lengthy operation. This is possible because the operation thread continues executing while the AVI clip is displayed. For example, the Find dialog box of Windows Explorer displays a moving magnifying glass as the system searches for

a file. An animation control can display an AVI clip originating from either an uncompressed AVI file or from an AVI file. You can add the AVI clip to your application as an AVI resource, or the clip can accompany your application as a separate AVI file.

---

## 8.8.1 ANIMATION CONTROL CREATION

---

An animation control belongs to the `ANIMATE_CLASS` window class. An animation control can be created by using the `CreateWindow` or `CreateWindowEx` function or the `Animate_Create` macro. The macro positions the animation control in the upper-left corner of the parent window and, if the `ACS_CENTER` style is not specified, sets the width and height of the control based on the dimensions of a frame in the AVI clip. If `ACS_CENTER` is specified, `Animate_Create` sets the width and height of the control to zero. You can use the `SetWindowPos` function to set the position and size of the control. If an animation control is created within a dialog box or from a dialog box resource, the control is automatically destroyed when the user closes the dialog box. If it is created within a window, control must be explicitly destroyed.

---

## 8.8.2 ABOUT ANIMATION CONTROL MESSAGES

---

An application sends messages to an animation control to open, play, stop, and close the corresponding AVI clip. Each message has one or more macros that can be used instead of sending the message explicitly. After creating an animation control, an application sends the `ACM_OPEN` message to open an AVI clip and load it into memory. The message specifies either the path of an AVI file or the name of an AVI resource. The system loads the AVI resource from the module that created the animation control.

If the animation control has the `ACS_AUTOPLAY` style, the control begins playing the AVI clip immediately after the AVI file or AVI resource is opened. Otherwise, an application can use the `ACM_PLAY` message to start the AVI clip. An application can stop the clip at any time by sending the `ACM_STOP` message. The last frame played remains displayed when the control finishes playing the AVI clip or when `ACM_STOP` is sent. An animation control can send two notification codes to its parent window: `ACN_START` and `ACN_STOP`. Most applications do not handle either notification. To close the AVI file or AVI resource and remove it from memory, an application can use the `Animate_Close` macro, which sends `ACM_OPEN` with the file name or resource name set to `NULL`.

### **CHECK YOUR PROGRESS**

- What do you mean by the `WM_PAINT` message?
- Discuss about animation control creation.
- Define `BeginPaint()` and `EndPaint()`.

---

## 8.9 SUMMARY

---

A graphic is an image or visual representation of an object. Therefore, computer graphics are simply images displayed on a computer screen. Graphics are often contrasted with text, which is comprised of characters, such as numbers and letters, rather than images. Computer graphics can be either two or three-dimensional. Early computers only supported 2D monochrome graphics, meaning they were black and white (or black and green, depending on the monitor). Eventually, computers began to support color images. While the first machines only supported 16 or 256 colors, most computers can now display graphics in millions of colors.

The Graphics Device Interface (GDI) is a Microsoft Windows application programming interface and core operating system component responsible for representing graphical objects and transmitting them to output devices such as monitors and printers. GDI is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes. A console application is a program which runs in a command prompt window. Console programs do not have the flash, nor the event-driven capabilities of a Windows application, however, they still have their place. For example, where screen space is limited such as a bank ATM, or a device with a very simple display type.

Multitasking, in an operating system, is allowing a user to perform more than one computer task (such as the operation of an application program) at a time. The operating system is able to keep track of where you are in these tasks and go from one to the other without losing information. Moreover, multitasking is the ability of an operating system to run various programs simultaneously. But actually at the background, OS uses a round-robin approach or a hardware clock i.e. it provides "time slices" for each running process; and a user thinks that all the programs are running concurrently. Multitasking is a very old concept which was present during the mainframe computers too. Then in those mainframes jobs were submitted and executed one-by-one accordingly. It took some time to become a reality or enter into personal computers.

Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. In a multithreaded environment, programs can split themselves into separate pieces (called threads of execution) that run concurrently. This was the best solution for the problem of queuing or serialization in the presentation manager. In terms of code, a thread is simply represented by a function which might also call other functions in the program.

The Clipboard is a shared memory area that you copy data into (aka. "copy", such as in response to the user pressing Ctrl+C) and copy data from (aka "paste"). The data can be simultaneously represented in dozens of common formats, and any number of programmer-defined formats.

The clipboard is a buffer that some operating systems provide for short-term storage and transfer within and between application programs. The clipboard is usually temporary and unnamed, and its contents reside in the computer's

RAM. The Clipboard is sometimes called the paste buffer. In Windows, but also in other operating systems like Android or Mac OS X, the clipboard is a special location in the PC or device memory, that is used as a temporary storage area for any information that is copied. Once some data is copied into the clipboard, it can be pasted somewhere else, in the same application from which it was copied, or in a different one, as long as it knows how to work with that type of data. It is also referred to as pasteboard, is a special location in computer's memory that temporarily stores data that was cut or copied from a document.

Printing from a Windows program usually involves some overhead, as well as some GDI calls to actually print something. The only experience you've had with GDI is the processing of the WM\_PAINT message in the main event handler. Remember that GDI, or the Graphics Device Interface, is how all graphics are drawn under Windows when DirectX is not in use. An animation control is a window that displays an Audio-Video Interleaved (AVI) clip. An AVI clip is a series of bitmap frames like a movie. Animation controls can only display AVI clips that do not contain audio. One common use for an animation control is to indicate system activity during a lengthy operation. This is possible because the operation thread continues executing while the AVI clip is displayed.

---

## 8.10 TERMINAL QUESTIONS

---

1. Define the term computer graphics and write the ways for creating graphics objects.
2. What is the basic difference between 2D graphics and 3D graphics?
3. Write a short note on consoles.
4. Compare the thread and process briefly.
5. Explain the concept and usefulness of multithreading.
6. What is event model? Draw its diagram.
7. Explain the process of drawing arcs and square.
8. Discuss the methods for brush class.
9. Explain the drag and drop operation in clipboard.
10. Write a short note on creating animation with picture clip control.



Uttar Pradesh Rajarshi Tandon  
Open University

Bachelor in Computer  
Application

**BCA-118**

**Windows Programming**

**BLOCK**

**4**

**INTERFACING AND DATABASE APPLICATION**

---

**UNIT-9**

**Interfacing Other Applications**

---

---

**UNIT-10**

**Database Application**

---

---

**UNIT-11**

**Network Programming**

---

---

**UNIT-12**

**Advanced Topics and Case Study**

---

---

## Course Design Committee

---

**Prof. Ashutosh Gupta**

Director

School of Science, UPRTOU Prayagraj

**Prof. Suneeta Agarwal**

Dept. of Computer Science & Engineering

Motilal Nehru National Institute of Technology, Allahabad, Prayagraj

**Dr. Upendra Nath Tripathi**

Associate Professor

Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur

**Dr. Ashish Khare**

Associate Professor

Dept. of Computer Science, University of Allahabad, Prayagraj

**Ms. Marisha**

Assistant Professor (Computer Science)

School of Science, UPRTOU Prayagraj

**Mr. Manoj Kumar Balwant**

Assistant Professor (Computer Science)

School of Sciences, UPRTOU Prayagraj

---

## Course Preparation Committee

---

**Dr. Krishan Kumar**

**Author**

Assistant Professor

Department of Computer Science,

Gurukula Kangri Vishwavidyalaya Haridwar (UK)

**Dr. Brajesh Kumar**

**Editor**

Associate Professor, Dept. of CS & IT

M.J.P Rohilkahand University, Bareilly, Uttar Pradesh

**Prof. Ashutosh Gupta**

**Director (In-Charge)**

School of Computer & Information Sciences

UPRTOU Prayagraj

**Mr. Manoj Kumar Balwant**

**Coordinator**

Assistant Professor (computer science)

School of Sciences, UPRTOU Prayagraj

---

©UPRTOU, Prayagraj - 2020

ISBN :

---

©All Rights are reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the **Uttar Pradesh Rajarshi Tondon Open University, Prayagraj.**

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.

**Printed By :** Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.



---

## BLOCK INTRODUCTION

---

*Block 4* includes four units i.e. unit 9, unit 10, unit 11, and unit 12. Unit 9 focusses on Interfacing Other Applications. Unit 10 is intended with detail description of DBMS. Unit 11 is related with network programming. And lastly unit 12 give an idea of how to develop a small Visual Basic application. At the end of this block, one would be able to understand about the following concepts:

- Exception handling
- OLE
- DBMS
- Database Access
- Creating tables, inserting, deleting and updating records
- VC++ resources
- Network programming with Windows Sockets, Securing Windows Objects
- COM
- DCOM
- An application using Visual Basic

*Unit 9* describes about the Single Document Interface (SDI) and Multiple Document Interface (MDI). Moreover, it explains about Splitter Windows, Exception Handling, Debugging, Object Linking and Embedding (OLE). Exception handling and debugging are two concepts which are very helpful in finding the error at runtime.

*Unit 10* mainly focusses on the Database Management System (DBMS). It explains about some other things associated to it like Odbc, DAO, Recordset etc. It also explains about the SQL (Structured Query Language) and its operations like creation of table, updation of table, deleting a record, performing different types of queries etc.

*Unit 11* mainly focusses on the Network programming concepts. It explains about network programming using Windows Sockets, Windows objects, and Securing Windows Objects.

*Unit 12* discusses about the ActiveX Controls, COM, DCOM, and COM+. Moreover, the development of an application using Visual Basic has been also given based on a case study.



## Structure

- 9.0 Introduction
- 9.1 Objectives
- 9.2 Single Document Interface (SDI)
- 9.3 Multiple Document Interface (MDI)
- 9.4 Difference Between SDI and MDI
- 9.5 Explorer Style-Interface
- 9.6 Splitter Windows
- 9.7 Exception Handling
- 9.8 Debugging
- 9.9 Object Linking and Embedding (OLE)
- 9.10 Summary
- 9.11 Terminal Questions

---

## 9.0 INTRODUCTION

---

A Windows Forms or WPF application provides several options for presenting graphic information. Deciding if the application needs to have a single document, multiple document, or navigation interface will affect the end user's satisfaction with the application's ability to meet his or her objective. A Windows Services application is designed to execute and interact with the Computer Management Console. The Windows Services application is intended to be executed for monitoring, maintaining, or evaluating functionality.

This Unit basically deals with document applications like single document interface (SDI), multiple document interface (MDI), splitter windows, exception handling, debugging, and object linking and embedding. SDI is intended with opening a single document at a time while on other hand various documents can be opened using MDI. Switching between various windows is easy in MDI. Splitter windows is another concept rather way to split the screen into two or more different panes. Splitting windows can be done using some derived objects. Splitter windows come in two forms - static and dynamic. This unit will only cover static splitters since the dynamic ones are slightly more complex. The main difference between the two is that a dynamic splitter can be split and collapsed by the user while a static splitter cannot.

This unit also explain about exception handling. An exception is an error condition, possibly outside the program's control, that prevents the program from continuing along its regular execution path. Certain operations, including object creation, file input/output, and function calls made from other modules, are all potential sources of exceptions even when the program is running correctly. Robust code anticipates and handles exceptions.

Without fail, the code written as a software developer, doesn't always do what expected it to do. Sometimes it does something completely different! When this happens, the next task is to figure out why, and although it might be tempted to just keep staring at code for hours, it's much easier and efficient to use a debugging tool, or debugger. A debugger, unfortunately, isn't something that can magically reveal all the problems or "bugs" in a code. *Debugging* means to run the code step by step in a debugging tool like Visual Studio, to find the exact point where one made a programming mistake. It is then understood what corrections are needed to make in code, and debugging tools often allow to make temporary changes so that the running program remain continue.

Finally, OLE that stands for "Object Linking and Embedding," has been discussed. OLE can be pronounced as "O-L-E," or "Oh-lay!" in Spanish. OLE is a framework developed by Microsoft (way back in Windows 3.1) that allows to take objects from a document in one application and place them in another. For example, OLE may allow to move an image from a photo-editing program into a word processing document. The OLE technology was initially created to allow the linking of objects between "compound documents," or documents that support multiple types of data. Microsoft has since developed OLE into a wider standard, known as the Component Object Model (COM). COM is supported by Mac, Unix, and Windows operating systems, but is primarily used with Microsoft Windows. The COM framework is the foundation of ActiveX, which allows developers to create interactive content for the Web.

Document/view applications aren't limited to just one document and one view of a document's data. MFC Document View Architecture can have two types of applications i.e. SDI and MDI. Using splitter windows provided by MFC, a single document interface (SDI) application can present two or more views of the same document in resizable "panes" that subdivide the frame window's client area. The document/view architecture also extends to multiple document interface (MDI) applications that support multiple views of a document, multiple open documents, and even multiple document types. Although Microsoft discourages the use of the multiple document interface, applications that rely on the MDI model are still prevalent and probably will be for some time to come, as evidenced by the continued success of Microsoft Word and other leading Microsoft Windows applications. MDI and SDI are different interface designs meant to handle documents within a single application. MDI allows an application to contain child windows per document, while SDI enforces one document per window.

Neither approach has much effect on performance or stability: despite the intuition of two SDI applications as being separate entities, they are very often still implemented as a single process. Similarly, an MDI interface can be implemented as multiple processes (Google Chrome being a prime example). It's

also worth noting that tabbed interfaces, although matching the description of MDI, often don't show multiple documents at the same time in the same window, and universally support multiple top-level windows as well.

---

## 9.1 OBJECTIVES

---

At the end of this unit you will come to know about:

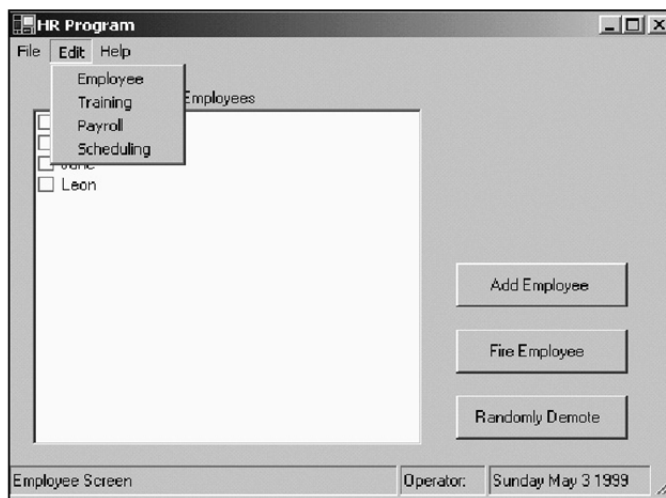
- Windows application types
- Windows user interface types
- Characteristics of Single Document Interface
- Role and applications of SDI
- Characteristics of Multiple Document Interface
- Difference between SDI and MDI
- Splitter windows
- Object Linking and Embedding
- Exception handling
- Debugging

---

## 9.2 SINGLE DOCUMENT INTERFACE

---

A Windows function that allows an application to display only one document at a time. The *Single Document Interface (SDI)* is one of the first UI designs introduced when the Windows operating system was created. SDI is a design pattern in which the graphical elements of the window apply only to the current Window where they reside (Fig. 9.1). The toolbars, menus, and other common Window elements control only the functionality for the window in which they are embedded. Each window that appears will have its own set of toolbars and menus to control its functionality.



**Fig. 9.1** Single Document Interface (SDI) example

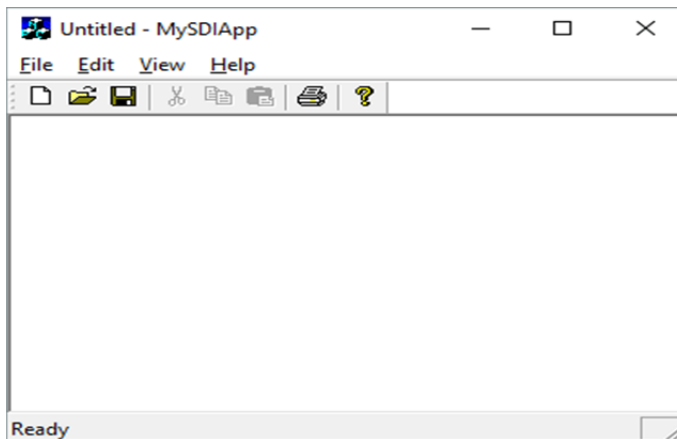
(Source: <http://icodeguru.com/dotnet/Data-Entry-and-Validation-with-CSharp-and-VB-Dot-NET-Windows-Forms/8214final/LiB0021.html>)

Moreover, SDI is a Single Document Interface as opposed to MDI which is a Multiple Document Interface; the difference is solely in the number of documents that can be opened at one time in each instance of the application. With SDI, only one document can be opened whereas with MDI many documents can be opened. This type of application can deal with a single document and single view of the document at one point of time. There is no way to open another document in the same application. Only way to open another document is to launch another instance of the application and open the another document. *Notepad*, *Wordpad* are such applications in Windows. This is the simplest interface one can present to the user. In a complicated program, however, it can also be the most confusing.

An SDI program requires that one screen must be closed before opening another. There would be no way to just flip back and forth between screens. This is the classic definition of an SDI program. Here are some common examples of SDI:

- *Notepad*: First document must be closed before editing another.
- *WordPad*: First document must be closed before editing another.
- *Calculator*: One calculator can be used at a time.

In a classic SDI program such as Notepad, if one wants to edit two documents at once he/she need to run two instances of the program. Neither instance of Notepad knows the other exists, and they do not interact. Fig 9.2 shows what a classic SDI HR application that includes an Employee screen might look like.



**Fig. 9.2 Classic SDI**

---

### **9.2.1 ANOTHER TYPE OF SDI PROGRAM**

---

Another type of SDI program is common these days. In the case of the hypothetical HR program, if the user chose the Training module while in the Employee screen, the program would not replace the Employee screen. It would instead bring up another form on the desktop, which would look like a separate

program altogether. However, it is not. The different screens can interact with each other and exchange information.

In this scenario, the user can have four forms open on the desktop, one for each module (Employee, Training, Payroll, and Scheduling). Can you imagine a program that could have a dozen or more forms open on the desktop at once? For anyone, this is chaos, but one already might have used a program that does this. VB 6.0 has an option to work in an SDI environment or an MDI environment. The default is MDI, and this keeps things organized. Word is arguably the most ubiquitous program in the world, and many people like to edit many documents at once. Word is now an SDI application, which means that each document is in a separate window.

---

## 9.2.2 THE MULTIFORM SDI EXAMPLE

---

Before the coding of an SDI program, one need to come up with a plan for how it will work. The normal working/plan for this one is based loosely on how Microsoft Word works:

- The program has a main form with a menu.
- The menu allows the user to choose different parts of the program to work on.
- Each part of the program is a new form that appears on the desktop in a random place.
- The main form has a Window menu option that shows all the HR screens the user has open.
- The Window option has a submenu item called "Close All Windows" that closes all the currently open windows.
- The Window menu option of the main screen denotes with a check mark the child form that is currently in focus.
- If the user chooses one of the forms listed under the Window menu option, that form gains focus.
- Only one instance of any of the forms can be running at a time.
- If the user chooses to edit a form that exists on the desktop, that form will gain focus.
- If the user closes an existing form, that form is deleted from the Window menu option.
- If the user closes the main form, the open windows automatically shut down.

Start a new C# or VB Windows project say "SDIFirst." One would need to follow the following steps before adding any code:

1. Add a MainMenu to the form.
2. Type File in the MainMenu item and call it *mnuFile*.
3. Below *mnuFile*, type in *Close* and call this item *mnuClose*.

4. Next to mnuFile, type in *Edit* and call this item *mnuEdit*.
5. Below mnuEdit, type in *Employee* and call this item *mnuEmp*.
6. Below mnuEmp, type in *Training* and call this item *mnuTrain*.
7. Below mnuTrain, type in *Payroll* and call this item *mnuPayRoll*.
8. Below mnuPayRoll, type in *Scheduling* and call this item *mnuSked*.
9. Next to mnuEdit, type in *Window* and call this item *mnuWindow*.
10. Next to mnuWindow, type in *Help* and call this item *mnuHelp*.
11. Add a status bar to the form.
12. Add a Panel to the Panels collection in the status bar and make it read *Employee Screen*. Change the AutoSize property to Spring.
13. Add a Panel to the Panels collection in the status bar and make it read *Operator*. Change the AutoSize property to Contents.
14. Add a Panel to the Panels collection in the status bar and type in the date. Change the AutoSize property to Contents.
15. Set the ShowPanels property to true.
16. Make the form start in the center of the screen.

---

## 9.3 MULTIPLE DOCUMENT INTERFACE

---

MDI stands for Multiple Document Interface. A user or programmer probably see many MDI applications many times. When multiple documents are handled, MDI forms are useful in a Windows program.

---

### 9.3.1 ADDING AN MDI FORM TO THE CURRENT PROJECT

---

Project -> Add MDI form. Click Project from the menu bar, and click Add MDI form. Remember, a project can have only one MDI form (Fig.9.3).

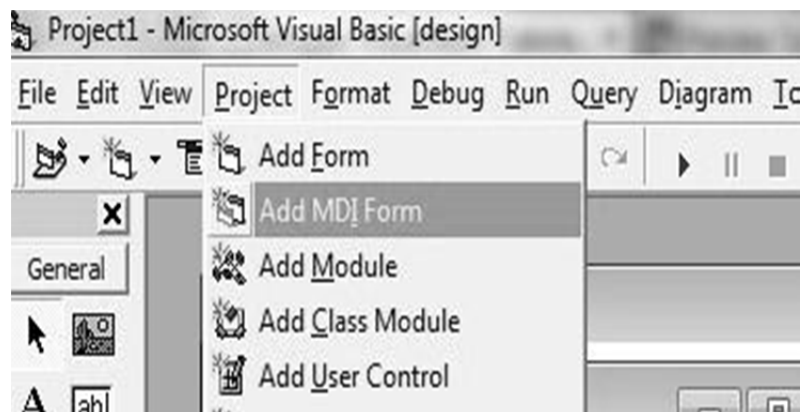


Fig. 9.3 Adding an MDI form



---

## 9.3.2 RESTRICTIONS OF THE MDI FORM

---

Usually there are two restrictions. These restrictions are there because MDI forms are the special type of forms, especially used to handle multiple child forms. These restrictions are:

- One MDI form per project can be created.
- Most controls on an MDI form cannot be placed.
- The only controls that can be placed on the surface of the MDI form are Menus, Timer, CommonDialog, PictureBox, ToolBar, and StatusBar.

---

## 9.3.3 WORKING OF THE MDI FORM

---

There can be only one MDI parent form in a project with one or more MDI child forms (or simply child forms).

---

### 9.3.3.1 MDI CHILD FORM

---

To add a child form, a regular form is added, and set the MDIchild property to True. You can have many child forms and can show an MDI child form using the Show method.

---

### 9.3.3.2 AUTOSHOWCHILDREN PROPERTY OF AN MDI FORM

---

The default value of the *AutoShowChildren* property is True. When it is True, the MDI child forms are displayed once they are loaded. When the value is False only then it can be kept hidden after loading, otherwise not.

---

### 9.3.3.3 RESTRICTIONS OF THE MDI CHILD FORMS

---

Normally there are two restrictions: you can't display an MDI child form outside its parent, and you can't display a menu bar on the MDI child form. Now coming to the point - how the MDI form works. The parent form contains a menu bar on top of it. From there, the user opens or creates a new document. In this way, the user accomplishes his/her work in one or multiple documents, then saves and closes the document (form). You can create instances of a single form in the code using the Set keyword (Using the object variables).

```
'Inside the MDIForm module  
Private Sub mnuFileNew_Click()  
    Dim frm As New Form1  
    frm.Show  
End Sub
```

---

### 9.3.3.4 ACTIVEFORM PROPERTY

---

This is the Object type read-only property of the MDI form. You can

apply this property to one of the children. For example, you can close the active form using this property from the Close menu command of the menu bar.

```
'In the MDI form
Private Sub mnuFileClose_Click()
    If Not (ActiveForm Is Nothing) Then Unload ActiveForm
End Sub
```

**Note:** - that '(ActiveForm Is Nothing)' represents that there is no active form. The 'Not' keyword before it negates the value.

---

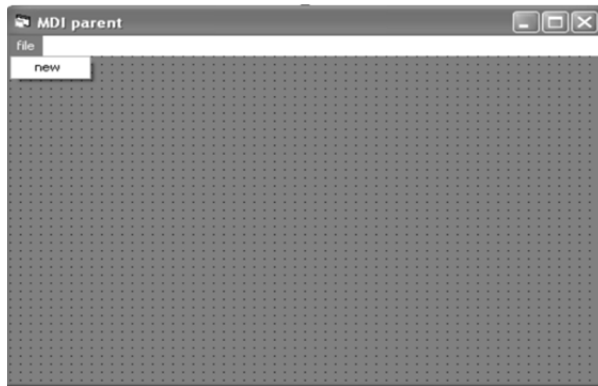
### 9.3.4 CREATING MDI APPLICATION USING VB

---

Following is an application program, that demonstrates about MDI application:

1. Start a new project by selecting file->new project. Select standard EXE as the project type if you have the project wizard enabled.
2. There will be already a form in the project. Set its name property to form child and its caption property to MDI child.
3. To create the MDI parent form (Fig. 9.4), right click the forms folder in the project Explorer and select add ->MDI form. If the form wizard appears, select MDI form.
4. Set the name property to formMDI and the caption property to MDI parent to MDI parent.
5. Right click project1 in the project Explorer and select project1 properties from the top-up menu. Set the startup object list to form MDI. If you omit this, the application will start with the child form (Fig. 9.5) showing.
6. Select form child from the project Explorer. Set the form's MDI child property to true. This will case this form, which is the child, to rest inside of the MDI parent container.
7. Select form MDI the project Explorer.
8. Start the menu designer by selecting tools->Menu Editor. You will see a window like the one in
9. Type & file in the caption field.
10. In the name field, type menufile.
11. Click the next button.
12. Click the arrow right button.
13. Enter & new in the caption field.
14. In the name field, type menunew.
15. Click the ok button to close the Menu Editor.
16. The form MDI from should now have a file menu on it. Select file ->New from the MDI menu. this will open up the window.

17. In the private sub menu file New-click () event
18. Save and Run the project.



**Fig. 9.4 Parent form**



**Fig. 9.5 Child Form**

**CHECK YOUR PROGRESS**

- What do you mean by SDI?
- Define about MDI.
- Give examples of SDI and MDI.

**9.4 DIFFERENCE BETWEEN SDI & MDI**

Features	MDI	SDI
Maximize all documents	Maximize parent window	Can only be implemented through special code or through a window manager that can group windows
Switch between documents	Using special interface inside parent window	Through task /window manager

Multiple Desktops	You can only stretch the parent window and try to organize individual windows manually	Easily done
Multiple Monitors	You can only span the parent window and try to organize individual windows manually	Easily done
Grouping	Naturally implemented	Possible only through special window managers

## 9.5 EXPLORER STYLE-INTERFACE

*File explorer or Microsoft repository browser*, previously known as *Windows Explorer*, is a file manager application that is included with releases of the Microsoft Windows operating system from Windows 95 onwards. This type of interface is very popular for navigating any type of dataset. The explorer has the several pieces of information that are persisted (Fig. 9.6). It provides a GUI for accessing the file systems. It is also the component of the operating system that presents many user interface items on the screen such as the taskbar and desktop. Controlling the computer is possible without Windows Explorer running (for example, the File | Run command in Task Manager on NT-derived versions of Windows will function without it, as will commands typed in a command prompt window).

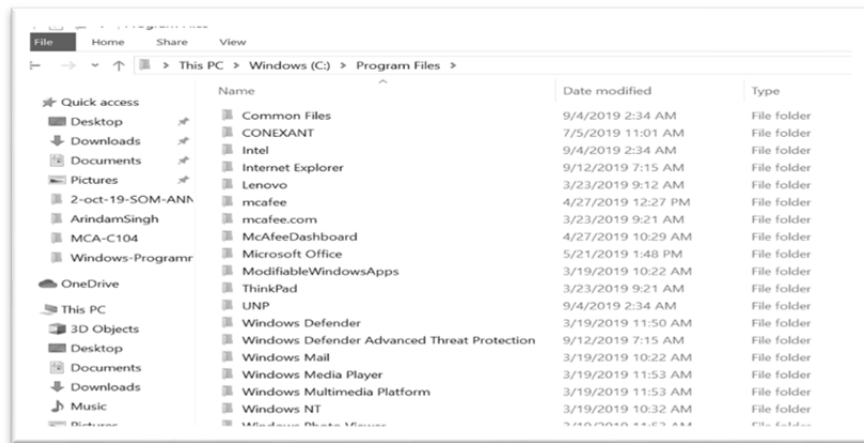


Fig. 9.6 Explorer Style Interface

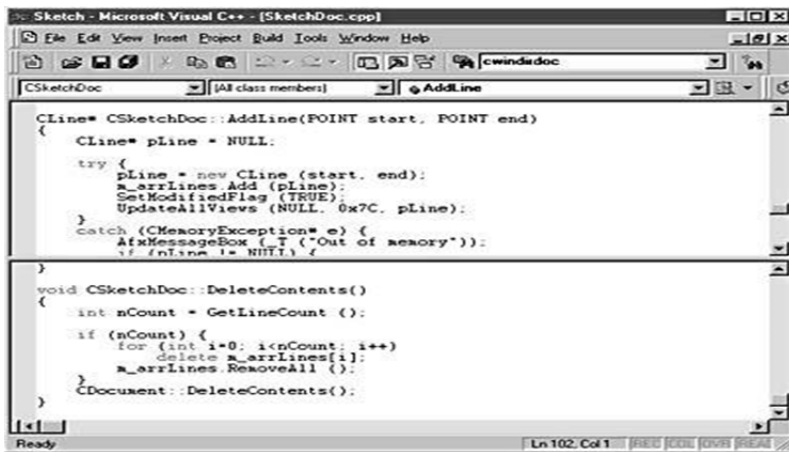
## 9.6 SPLITTER WINDOWS

MDI applications inherently support multiple views of a document; SDI applications do not. For SDI applications, the best way to present two or more concurrent views of a document is to use a splitter window based on MFC's *CSplitterWnd* class. A splitter window is a window that can be divided into two or

more panes horizontally, vertically, or both horizontally and vertically using movable splitter bars (Fig. 9.6). Each pane contains one view of a document's data. The views are children of the splitter window, and the splitter window itself is normally a child of a frame window. In an SDI application, the splitter window is a child of the top-level frame window. In an MDI application, the splitter window is a child of an MDI document frame. A view positioned inside a splitter window can use `CView::GetParentFrame` to obtain a pointer to its parent frame window.

MFC supports two types of splitter windows: *static* and *dynamic*. The numbers of rows and columns in a static splitter window are set when the splitter is created and can't be changed by the user. The user is, however, free to resize individual rows and columns. A static splitter window can contain a maximum of 16 rows and 16 columns. For an example of an application that uses a static splitter, look no further than the Windows Explorer. Explorer's main window is divided in half vertically by a static splitter window.

A dynamic splitter window is limited to at most two rows and two columns, but it can be split and unsplit interactively. The views displayed in a dynamic splitter window's panes aren't entirely independent of each other: when a dynamic splitter window is split horizontally, the two rows have independent vertical scroll bars but share a horizontal scroll bar. Similarly, the two columns of a dynamic splitter window split vertically contain horizontal scroll bars of their own but share a vertical scroll bar. The maximum number of rows and columns a dynamic splitter window can be divided into are specified when the splitter is created. Thus, it's a simple matter to create a dynamic splitter window that can be split horizontally or vertically but not both. Visual C++ uses a dynamic splitter window to permit two or more sections of a source code file to be edited at once.



**Fig. 9.6 A dynamic splitter showing two views of a document in Visual C++**

One criterion for choosing between static and dynamic splitter windows is whether you want the user to be able to change the splitter's row and column configuration interactively. Use a dynamic splitter window if you do. Another factor in the decision is what kinds of views you plan to use in the splitter's panes. It's easy to use two or more different view classes in a static splitter window

because you specify the type of view that goes in each pane. MFC manages the views in a dynamic splitter window, however, so a dynamic splitter uses the same view class for all of its views unless you derive a new class from *CSplitterWnd* and modify the splitter's default behavior.

---

## 9.6.1 DYNAMIC SPLITTER WINDOWS

---

Dynamic splitter windows are created with MFC's *CSplitterWnd::Create* function. Creating and initializing a dynamic splitter window is a simple two-step procedure:

- Add a *CSplitterWnd* data member to the frame window class.
- Override the frame window's virtual *OnCreateClient* function, and call *CSplitterWnd::Create* to create a dynamic splitter window in the frame window's client area.

Assuming *m\_wndSplitter* is a *CSplitterWnd* object that's a member of the frame window class *CMainFrame*, the following *OnCreateClient* override creates a dynamic splitter window inside the frame window:

```
BOOL CMainFrame::OnCreateClient (LPCREATESTRUCT lpcs,
                                CCreateContext* pContext){return
    m_wndSplitter.Create(this, 2, 1, CSize(1,1), pContext); }
```

The first parameter to *CSplitterWnd::Create* identifies the splitter window's parent, which is the frame window. The second and third parameters specify the maximum number of rows and columns that the window can be split into. Because a dynamic splitter window supports a maximum of two rows and two columns, these parameter values will always be 1 or 2. The fourth parameter specifies each pane's minimum width and height in pixels. The framework uses these values to determine when panes should be created and destroyed as splitter bars are moved. *CSize* values equal to (1,1) specify that panes can be as little as 1 pixel wide and 1 pixel tall. The fifth parameter is a pointer to a *CCreateContext* structure provided by the framework. The structure's *m\_pNewViewClass* member identifies the view class used to create views in the splitter's panes. The framework creates the initial view for you and puts it into the first pane. Other views of the same class are created automatically as additional panes are created.

*CSplitterWnd::Create* supports optional sixth and seventh parameters specifying the splitter window's style and its child window ID. In most instances, the defaults are fine. The default child window ID of *AFX\_IDW\_PANE\_FIRST* is a magic number that enables a frame window to identify the splitter window associated with it. You need to modify the ID only if you create a second splitter window in a frame window that already contains a splitter.

Once a dynamic splitter window is created, the framework provides the logic to make it work. If the window is initially unsplit and the user drags a vertical splitter bar to the middle of the window, for example, MFC splits the window vertically and creates a view inside the new pane. Because the new view is created at run time, the view class must support dynamic creation. If the user

later drags the vertical splitter bar to the left or right edge of the window (or close enough to the edge that either pane's width is less than the minimum width specified when the splitter window was created), MFC destroys the secondary pane and the view that appears inside it.

The *CSplitterWnd* class includes a number of useful member functions you can call on to query a splitter window for information. Among other things, you can ask for the number of rows or columns currently displayed, for the width or height of a row or a column, or for a *CView* pointer to the view in a particular row and column. If you'd like to add a Split command to your application's menu, include a menu item whose ID is `ID_WINDOW_SPLIT`. This ID is prewired to the command handler *CView::OnSplitCmd* and the update handler *CView::OnUpdateSplitCmd* in *CView*'s message map. Internally, *CView::OnSplitCmd* calls *CSplitterWnd::DoKeyboardSplit* to begin a tracking process that allows phantom splitter bars to be moved with the up and down arrow keys. Tracking ends when Enter is pressed to accept the new splitter position or Esc is pressed to cancel the operation.

### **CHECK YOUR PROGRESS**

- What is the main difference between SDI and MDI?
- Is there any difference between file explorer and the window explorer?
- Describe the meaning of splitting windows.

---

## **9.7 EXCEPTION HANDLING**

---

An *exception* is an event that occurs during the execution of a program, and requires the execution of code outside the normal flow of control. There are two kinds of exceptions: *hardware exceptions* and *software exceptions*. *Hardware exceptions* are initiated by the CPU and they can result from the execution of certain instruction sequences, such as division by zero or an attempt to access an invalid memory address. *Software exceptions* are initiated explicitly by applications or the operating system. For example, the system can detect when an invalid parameter value is specified. There are some types of exceptions, each with their own way of dealing with them. First, there are Programming and Environmental errors returned from INtime system calls, defined by status codes. Then there are Numerics exceptions as a result of numerical operations. Hardware faults are caused by using hardware resources incorrectly. And then there are exceptions handled by Structured Exception Handling and C++ exception handling.

*Structured exception handling* is a mechanism for handling both hardware and software exceptions. Therefore, your code will handle hardware and software exceptions identically. Structured exception handling enables you to have complete control over the handling of exceptions, provides support for debuggers, and is usable across all programming languages and machines. *Vectored*

*exception handling* is an extension to structured exception handling. The system also supports *termination handling*, which enables you to ensure that whenever a guarded body of code is executed, a specified block of termination code is also executed. The termination code is executed regardless of how the flow of control leaves the guarded body. For example, a termination handler can guarantee that clean-up tasks are performed even if an exception or some other error occurs while the guarded body of code is being executed.

---

## 9.7.1 PROGRAMMING AND ENVIRONMENTAL EXCEPTIONS

---

Programmer exceptions are caused by providing incorrect parameters to a system call. Environmental exceptions occur when the environment cannot provide the requested resource, such as trying to allocate memory when none is available. Programming and environmental exceptions are always indicated to the calling thread by a specific return value from the INtime system call; the actual return values for success and exceptions are dependent on the type of system call made. When an exception is indicated, you can call `GetLastRtError` to retrieve the status code; you may also want your application to investigate the system's state by using calls that return system accounting information. If a system call returns a result that indicates an exception, all other output parameters of that system call are undefined.

---

## 9.7.2 STRUCTURED EXCEPTION HANDLING

---

Structured Exception handling is a Microsoft specific extension to the C programming language, implemented by Microsoft Visual Studio and supported by the operating system, in particular by Windows and also by INtime. You can use Structured Exception Handling in C source code to deal with Hardware faults and with user defined exceptions.

Using `__try` and `__except` blocks and some APIs like `GetExceptionCode` and `RaiseException`, you can decide how a thread reacts to hardware faults and user exceptions. An example follows:

```
/* a filter indicates if the __except clause handles this
type of exception */
/* this filter handles all exceptions */
int Filter(LPEXCEPTION_POINTERS pEP) {
    printf("Exception %x at address %x\n",
        pEP->ExceptionRecord.ExceptionCode,
        pEP->ExceptionRecord.ExceptionAddress);
    return EXCEPTION_EXECUTE_HANDLER;
}

__try {
    /* perform some action that may cause a hardware
exception */
```



```

}
__except(Filter(GetExceptionInformation())) {
    ExitProcess(1);
}

```

### **Example**

```

Module Module1
Sub Main()
Dim i=0, j=1, k As Integer
try
k=j/i
System.Console.WriteLine("The result is " & k)
Catch e As Exception
System.Console.WriteLine(e.Message)
End Try
End Sub
End Module

```

The result of this sample code is that if the action does indeed cause a hardware exception, the fault code and address are printed in the console window and the process is terminated. For more details, see `__try`, `__except` and `__finally` in the Microsoft Visual Studio documentation. In the filter expression (the expression in parentheses following the `__except` keyword) you can use the *GetExceptionCode* and *GetExceptionInformation* functions. They return the exception code and an EXCEPTION\_POINTERS pointer that provide details on the exception. The filter expression indicated if the `__except` clause is executed to handle the exception or not. If you wish to force an exception that is handled like a hardware fault, you can use the *RaiseException* function.

---

#### **9.7.2.1 FEATURES OF STRUCTURED EXCEPTION HANDLING**

---

- Code is easy to read, debug and maintain
- Allows to create protected blocks of code
- Allows nested Handling
- Allows filtering of exceptions similar to select case statement

---

#### **9.7.3 UNSTRUCTURED EXCEPTION HANDLING**

---

- Code is difficult to read, debug and maintain
- Errors may be overlooked

#### **Syntax:**

On Error GoTo [line|0|-1] Resume Next

- *GoTo Line*- Enables the exception handling code that starts at the line specified in the required line argument. The line argument is any line number or label. The specified line must be in the same procedure.
- *GoTo 0*- Disables enabled exception handler in the current procedure and resets to nothing.
- *GoTo -1*- same above
- *Resume Next*- Specifies that when an exception occurs, execution skips over the statement that caused the problem and goes to the statement immediately following. Execution continues from that point

### **Example**

```

Module Module1
Sub Main()
Dim i=0, j=1, k As Integer
On Error GoTo Handler
k=j/i
Exit Sub

Handler:
System.Console.WriteLine("Divide By Zero")

End Sub

End Module

```

---

## **9.8 DEBUGGING**

---

Debugging is the routine process of locating and removing computer program bugs, errors or abnormalities, which is methodically handled by software programmers via debugging tools. Debugging checks, detects and corrects errors or bugs to allow proper program operation according to set specifications. Debugging is also known as debug.

While, a software bug is a problem causing a program to crash or produce invalid output. The problem is caused by insufficient or erroneous logic. A bug can be an error, mistake, defect or fault, which may cause failure or deviation from expected results. Most bugs are due to human errors in source code or its design. A program is said to be buggy when it contains a large number of bugs, which affect program functionality and cause incorrect results.

Some bugs might not have serious effects on the functionality of the program and may remain undetected for a long time. A program might crash when serious bugs are left unidentified. Another category of bugs called security bugs may allow a malicious user bypass access controls and obtain unauthorized privileges.

Some of the worst bugs in history include:

- In the 1980s, bugs in the code controlling the machine called *Therac-25*, used for radiation therapy, lead to patient deaths.

- In 1996, the \$1.0 billion rocket called *Ariane 5* was destroyed a few seconds after launch due to a bug in the on-board guidance computer program.
- In 1962, a bug in the flight software for the *Mariner I* spacecraft caused the rocket to change path from the expected path.
- In the 1990s, a bug was found in the new release of AT&T's software control #4ESS long distance switches caused many computers to crash.

Developing software programs undergo heavy testing, updating, troubleshooting and maintenance. Normally, software contains errors and bugs, which are routinely removed. In the debugging process, complete software programs are regularly compiled and executed to identify and rectify issues. Large software programs, which contain millions of source code lines, are divided into small components. For efficiency, each component is debugged separately at first, followed by the program as a whole.

Debugging, in computer programming and engineering, is a multistep process that involves identifying a problem, isolating the source of the problem, and then either correcting the problem or determining a way to work around it. The final step of debugging is to test the correction or workaround and make sure it works.

---

## 9.8.1 THE DEBUGGING PROCESS

---

In software development, debugging involves locating and correcting *code errors* in a computer program. Debugging is part of the software testing process and is an integral part of the entire software development lifecycle. The debugging process starts as soon as code is written and continues in successive stages as code is combined with other units of programming to form a software product. In a large program that has thousands of lines of code, the debugging process can be made easier by using strategies such as *unit tests*, *code reviews* and *pair programming*.

Once an error has been identified, it is necessary to actually find the error in the code. At this point, it can be useful to look at the code's logging and use a stand-alone debugger tool or the debugging component of an integrated development environment (IDE). Invariably, the bugs in the functions that get most use are found and fixed first. In some cases, the module that presents the problem is obvious, while the line of code itself is not. In that case, unit tests such as *JUnit* and *xUnit*, which allow the programmer to run a specific function with specific inputs can be helpful in debugging.

The standard practice is to set up a "breakpoint" and run the program until that breakpoint, at which time program execution stops. The debugging component of an IDE typically provides the programmer with the capability to view memory and see variables, run the program to the next breakpoint, execute just the next line of code, and, in some cases, change the value of variables or even change the contents of the line of code about to be executed.

---

## 9.8.2 COMMON DEBUGGING TOOLS

---

Source code analyzers, which include security, common code errors and complexity analyzers, can also be helpful in debugging. A complexity analyzer can find modules that are so typical as to be hard to understand and test. Some tools can actually analyze a test run to see what lines of code are not executed, which can aid in debugging. Other debugging tools include advanced logging and simulators that allow the programmer to model how an app on a mobile device will display and behave.

Some tools, especially open source tools and scripting languages, do not run in an IDE and require a more manual approach to debugging. Such techniques include dropping values to a log, extensive "print" statements added during code execution or hard-coded "wait" commands that simulate a breakpoint by waiting for keyboard input at specific times.

The use of the word *bug* as a synonym for *error* originated in engineering. The term's application to computing and the inspiration for using the word *debugging* as a synonym for *troubleshooting* has been attributed to *Admiral Grace Hopper*, a pioneer in computer programming, who was also known for her dry sense of humor. When an actual bug (a moth) got caught between electrical relays and caused a problem in the U.S. Navy's first computer, Admiral Hopper and her team "debugged" the computer and saved the moth. It now resides in the Smithsonian Museum.

---

## 9.9 OBJECT LINKING AND EMBEDDING

---

OLE stands for "Object Linking and Embedding." It can be pronounced as "O-L-E," or "Oh-lay!" if you are feeling Spanish. OLE is a framework developed by Microsoft that allows to take objects from a document in one application and place them in another. For example, OLE may allow to move an image from a photo-editing program into a word processing document.

OLE Server is an application that can provide objects to other applications. This is also called as OLE Source application. OLE Client is an application that uses objects provided by OLE Server. This is also called as OLE Container as it contains objects provided by OLE Server. Not every application is an OLE Server. Only a few applications are capable of providing objects. In the same way not all applications are capable of receiving objects. However, there are applications, such as MS Word and MS Excel that are capable of being OLE source as well as OLE Container.

The OLE technology was initially created to allow the linking of objects between "compound documents," or documents that support multiple types of data. Microsoft has since developed OLE into a wider standard, known as the Component Object Model (COM). COM is supported by Mac, Unix, and Windows systems, but is primarily used with Microsoft Windows. The COM framework is the foundation of ActiveX, which allows developers to create interactive content for the Web.

*Object Linking & Embedding (OLE)* is a technology developed by

Microsoft that allows embedding and linking to documents and other objects. For developers, it brought OLE Control Extension (OCX), a way to develop and use custom user interface elements. On a technical level, an OLE object is any object that implements the IOleObject interface, possibly along with a wide range of other interfaces, depending on the object's needs.

OLE allows an editing application to export part of a document to another editing application and then import it with additional content. For example, a desktop publishing system might send some text to a word processor or a picture to a bitmap editor using OLE. The main benefit of OLE is to add different kinds of data to a document from different applications, like a text editor and an image editor. This creates a Compound File Binary Format document and a master file to which the document makes reference. Changes to data in the master file immediately affect the document that references it. This is called "linking" (instead of "embedding"). OLE is also used for transferring data between different applications using drag and drop and clipboard operations.

One of the key point is that OLE transformed software development from procedural programming languages to *object-oriented programming*. We can create self-contained Modules, or objects with the help of OLE, that simplify programming approach to building large applications. OLE Means that objects created from different formats that can be linked and embedding application data.

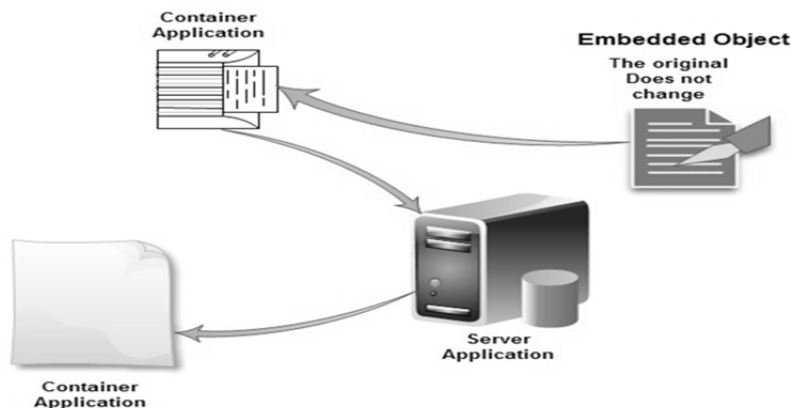
**Example:** The basic example of OLE is that, when we can insert Excel spreadsheet into a Word application.

---

### 9.9.1 OLE EMBEDDING MEANING

---

The Definition of Embedding means is that if One window application document contains a copy of other window application document, if changes made that affect only the application document that contains it.



**Fig. 9.7 OLE Embedding**

(Source: <https://networkencyclopedia.com/object-linking-and-embedding-ole/>)

An object might be a passage of formatted text, a part of a spreadsheet, some sounds, or a picture. Unlike information that is copied from one document

and pasted into another the standard way, a linked or embedded object retains a connection to the application that originally created it. It can be returned to that application to edit the object whenever you want to just by double-clicking on the object-you don't have to bother with finding the icon for the application, loading the right file, and so on. Better yet, the changes you make automatically appear in the document where you linked or embedded the object. When you embed an object, you place a copy of the information into your document. This copy is connected to the original application, but not to a particular document in that application. The only advantage to embedding an object instead of copying the information the ordinary way is that you can edit the object more conveniently.

By contrast, when you link an object, you place a "reference" to a particular document from another application into the document you're working with. Let's say you have a spreadsheet that totals your third quarter sales figures. You link that spreadsheet document into a report you're preparing in your word processor. Later, when revised sales figures come in, you go back to the spreadsheet application and change the numbers. The next time you open the report document in your word processor, the new figures from the spreadsheet appear automatically in the report. This is the same idea as a "hot link," and it may help to read the generic definition for link. OLE only works if both applications involved have been designed to use it, and even then it may only work in one direction (like, you can link a graphic into a text document, but not text into a graphic document). And it doesn't work exactly the same way in every application. Even so, it's easier and more consistent than the old method, called DDE.

---

## **9.9.2 ADVANTAGES AND DISADVANTAGES OF OLE EMBEDDING**

---

Object linking and embedding (OLE) refers to the practice of providing a link in a document directly to a source of data or a graphic or embedding that data or graphic in the document itself. OLE's features make it a convenient tool for those creating documents or presentations that need to be kept up to date, but it has some downsides as well. The advantage with Object Embedding is, client application maintains its own copy of the data. The disadvantage is, changes made to original data (in source application) will not be incorporated in the data maintained by client. Some of advantages and disadvantages are described below.

---

### **9.9.2.1 CONTROLLABLE INFORMATION**

---

You can maintain control over the source through object linking. The link goes back to information that you can control, so you can quickly and conveniently update the information or graphic without needing to point the user to a new source. A person can go back and revisit the link over and over again to get the new information.

---

### **9.9.2.2 CONVENIENCE**

---

An embedded file can be quite convenient for users of the presentation or

document, as they will be able to view the file or graphic right in the document without having to click through a link or follow a web address, which may require the user to log in first or jump through other hurdles.

---

### **9.9.2.3 RESTRICTED ACCESS**

---

All users have to have access to the file and the application that runs it, which may prove to be a disadvantage if you have people who need to access the link who don't have the right permissions or who aren't able to install the correct program. In that case, your presentation or document is only as good as the privileges or software that your users have. This is not as big of a problem if you have a lot of users who are on the same network or work in the same office.

---

### **9.9.2.4 EMBED PROBLEMS**

---

An embedded file will show up as just a snapshot or it won't be displayed at all if a user can't obtain access to the file. This can derail a critical presentation or document that depends on the information you are embedding within it. You may have to test the embed from the systems that will need to reach it to ensure that it works, which may take a lot of time to do.

---

## **9.9.3 EXAMPLE OF OLE**

---

The following examples, where we embed a few cells of MS Excel spreadsheet to a MS Word document, will make this process clear:

- A collection of cells from a spreadsheet of MS Excel is copied to clipboard. As MS Excel is an OLE Server, it copies the data in the form of an object.
- Paste the data (now in the form of an object) from Clipboard to a document in MS Word.
- Now the data is embedded into MS Word document as an object. MS Word document contains its own copy of the data.
- If you double click on the object in MS Word, then an instance of MS Excel is invoked and data from MS Word is copied into MS Excel.
- User can edit embedded data using MS Excel.
- If user saves changes and exits MS Excel, then modified data is placed in MS Word document.

---

## **9.9.4 CREATING OLE CONTAINER CONTROL**

---

To create OLE Container control:

- Select OLE control in Toolbox.
- Place the control on the form with the required size.

- As soon as OLE control is placed on the form, Insert Object Dialog is displayed to allow you to either embed or link an object into OLE Container.
- The available options in insert dialog are - Create New, where you select an Object Type and create an object using the appropriate application, or Create from File, where you can create an object by selecting a file from file system.

---

### 9.9.5 EMBEDDING A WORD DOCUMENT INTO OLE CONTAINER CONTROL

---

To embed a word document into OLE Container control:

- In Insert Object Dialog box select Create from File radio button.
- Click on Browse button and select a document file.
- Click on Ok
- An object is embedded into OLE Container control and a part of document is displayed.
- Run the project using F5.
- Double click on OLE Container control. This action will invoke MS Word and run it in OLE Container control. When OLE Server runs in OLE Client, it is called as In-Place Activation.
- Make necessary changes using MS Word.
- Press ESC key to come out of In-Place activation.

**Note:** - *When you activate object, If OLE Server runs in client application, it is called as In-place Activation. A word document in OLE container on Visual Basic form at runtime.*

#### **CHECK YOUR PROGRESS**

- What do you mean by exception?
- Compare hardware exception and software exception.
- Write one advantage and disadvantage of OLE embedding.

---

### 9.10 SUMMARY

---

*Single Document Interface (SDI)* is a design pattern in which the graphical elements of the window apply only to the current window where they reside. The toolbars, menus, and other common window elements control only the functionality for the window in which they are embedded. Each window that appears will have its own set of toolbars and menus to control its functionality.



MDI stands for *Multiple Document Interface*. A user or programmer probably see many MDI applications many times. When multiple documents are handled, MDI forms are useful in a Windows program.

*File Explorer*, previously known as *Windows Explorer*, is a file manager application that is included with releases of the Microsoft Windows operating system from Windows 95 onwards. It provides a GUI for accessing the file systems. It is also the component of the operating system that presents many user interface items on the screen such as the taskbar and desktop.

A *Splitter window* is a window that can be divided into two or more panes horizontally, vertically, or both horizontally and vertically using movable splitter bars. Each pane contains one view of a document's data. The views are children of the splitter window, and the splitter window itself is normally a child of a frame window. In an SDI application, the splitter window is a child of the top-level frame window. In an MDI application, the splitter window is a child of an MDI document frame.

An *exception* is an event that occurs during the execution of a program, and requires the execution of code outside the normal flow of control. There are two kinds of exceptions: hardware exceptions and software exceptions. *Hardware exceptions* are initiated by the CPU. They can result from the execution of certain instruction sequences, such as division by zero or an attempt to access an invalid memory address. *Software exceptions* are initiated explicitly by applications or the operating system.

*Structured Exception Handling* is a Microsoft specific extension to the C programming language, implemented by Microsoft Visual Studio and supported by the operating system, in particular by Windows and also by INtime. You can use Structured Exception Handling in C source code to deal with Hardware faults and with user defined exceptions.

*Debugging* is the routine process of locating and removing computer program bugs, errors or abnormalities, which is methodically handled by software programmers via debugging tools. Debugging checks, detects and corrects errors or bugs to allow proper program operation according to set specifications. *OLE* stands for "Object Linking and Embedding." It can be pronounced as "O-L-E," or "Oh-lay!" if you are feeling Spanish. OLE is a framework developed by Microsoft that allows to take objects from a document in one application and place them in another. For example, OLE may allow to move an image from a photo-editing program into a word processing document.

---

## 9.11 TERMINAL QUESTIONS

---

1. Describe SDI and MDI using examples.
2. Write the steps to create an MDI application.
3. Compare SDI and MDI.
4. Write a short note on explorer style interface.
5. Explain splitter windows in detail.

6. Write a short note on exception handling.
7. Differentiate structured and unstructured exception handling.
8. Define the term debugging and also discuss the debugging process.
9. Explain the merit and demerits of OLE.
10. What do you understand by OLE? How can it be created?

## Structure

- 10.0 Introduction
- 10.1 Objectives
- 10.2 History of DBMS
- 10.3 Introduction to DBMS
- 10.4 DBMS Architecture
- 10.5 Components of DBMS
- 10.6 Need of DBMS
- 10.7 Advantages of DBMS
- 10.8 Disadvantages of DBMS
- 10.9 Database Administrator (DBA)
- 10.10 Open Database Connectivity (ODBC)
- 10.11 Database Access
- 10.12 Structured Query Language (SQL)
- 10.13 Database Access with Data Control
- 10.14 Recordset
- 10.15 Applications of DBMS
- 10.16 Summary
- 10.17 Terminal Questions

---

## 10.0 INTRODUCTION

---

A Database Management System or DBMS in short, is a software used to store and manage data. The DBMS was introduced during 1960's to store any data. It also offers manipulation of the data like insertion, deletion, and updating of the data. DBMS is a very important part of computers. Technically, it is a collection of interrelated data which exists in the records may be in form of like tables etc. More specifically it is a set of programs used to access the data from the database. DBMS basically contains the information of a particular organization/enterprise. It is the starting point from where generally a software starts its journey during the time of development. As, it should be very clear for a developer about the actual facts of the enterprise (s) which are normally available in some well-defined form which is a database. And the computer program/software is called the DBMS.

This unit is an introduction to the design, use, and internal workings of DBMS. Moreover, it deals with the DBMS's definition, main objectives, functions, advantages, and applications. The unit also describes about the database manager, ODBC, database access. Moreover, it discusses about the most commonly used controls like ADO, recordset etc. To understand this chapter some prerequisites are also necessary. These are like – Data Structures, Discrete Structures, Structured Programming Language (C++), Software engineering topics related to project documentation and project design

Databases touch all aspects of our lives, one cannot avoid it rather it can be said that it is everywhere. Few of its applications are Banking sector, Airlines, Education, Sales, Manufacturing, Human resource, Social Security Info, Online shopping etc. You can say it actually makes the current society and your life work. DBMS system also performs the functions like defining, creating, revising and controlling the database. It is specially designed to create and maintain data and enable the individual business application to extract the desired data.

As far as DBMS marketplace is concerned; Relational DBMS companies – Oracle, Sybase – are among the largest software companies in the world. IBM offers its relational DB2 system. With IMS, a nonrelational system, IBM is by some accounts the largest DBMS vendor in the world. Microsoft offers SQL-Server, plus Microsoft Access for the cheap DBMS on the desktop, answered by “lite” systems from other competitors. OpenSource: MySQL, PostgreSQL.

---

## 10.1 OBJECTIVES

---

At the end of this unit you will come to know about

- Evolution of DBMS
- Difference between the File Processing System and DBMS
- Characteristics of DBMS
- Need of DBMS
- Applications of DBMS
- SQL
- Recordset
- DBA
- Role and applications of DBMS
- DBMS vs RDBMS

---

## 10.2 HISTROY OF DBMS

---

**1960's-1970's-** The emergence of the first type of DBMS, the hierarchical DBMS. IBM had the first model, developed on IBM 360 and their (DBMS) was called IMS, originally it was written for the Apollo program. This type of DBMS was based on binary trees, where the shape was like a tree and relations were only limited between parent and child records. The benefits were numerous; less redundant data, data independence, security and integrity, which all lead to efficient searches. Nonetheless; there were some disadvantages such as; complex implementation, was hard to manage because of the absence of standards, which made it harder to handle many relationships.

**1960's-1970's-** The emergence of the network DBMS. Charles Bachmann developed first DBMS at Honeywell, Integrated Data Store(IDS) then a group called CODASYL who is responsible for the creation of COBOL, had that system standardized. However; the CODASYL group invented what they call the "CODASYL APPROACH." Based on that approach many systems using network DBMS were developed for business use. In this model, each record can have multiple parents in comparison with one in the hierarchical DBMS. It is made of sets of relationships where a set represents a one to many relationships between the owner and the member. The main and unfortunate disadvantage was that the System was complex and there was difficulty in design and maintenance, it is believed that the Lack of structural independence was the main cause.

**1970's- 1990's-** The emergence of the relational DBMS on the hands of Edgar Codd. He worked at IBM, and he was unhappy with the navigational model of the CODASYL APPROACH. To him, a tool for searching, such as a search facility was very useful, and it was absent. In 1970, he proposed a new approach to database construction, which made the creation of a Relational DBMS intended for Large Shared Data Banks, possible and easy to grab. Moreover; this was a new system for entering data and working with big databases, where the idea was to use a table of records. All tables would be then linked by either one-to-one relationships, one-to-many, or many-to-many. When elements took space and were not useful, it was easy to remove them from the original table, and all the other "entries" in other tables linked to this record were removed.

Worth mentioning, is that two initial projects were launched, the R program at IBM, and INGRES program at the University of California. In 1985, the object oriented DBMS was developed, but it did not have any booming commercial profit because of the high unjustified costs to change systems, and format. In 1990, the DBMS took on a new object oriented approach joint with relational DBMS. In this approach, text, multimedia, Internet and web use in conjunction with DBMS were available and possible. In the early years of computing, a punch card was used in unit record machines for input, data storage and processing this data. Data was entered offline and for both data, and computer programs input. This input method is similar to voting machines nowadays. This was the only method, where it was fast to enter data, and retrieve it, but not to manipulate or edit it. After that era, there was the introduction of the file type entries for data, then the DBMS as hierarchical model, network model, and relational model.

---

## **10.3 INTRODUCTION OF DBMS**

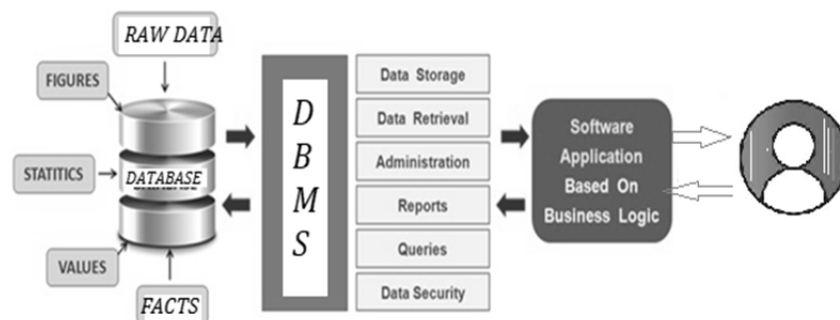
---

Before knowing about the DBMS, it is necessary to understand about database. A database is an organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex than they are often developed using formal design and modeling techniques. By data, we mean known facts that can be recorded and have embedded meaning too. Usually people use software such as Oracle, MS Access, or MS Excel to store data in the form of databases. A datum is a unit of data. Meaningful data combined to form information. Hence, information is basically interpreted data - data provided with semantics.

Knowledge refers to the useful use of information. As you know, that information can be transported, stored and shared without much problem and difficulties but the same cannot be said about knowledge. Knowledge necessarily involves a personal experience and practice. Database systems are meant to handle a large collection of information. Management of data involves both defining structures for storage of information and providing mechanisms that can do the manipulation of those stored information. Moreover, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.

A DBMS refers to the technology for creating and managing databases. DBMS is a software tool to organize (create, retrieve, update and manage) data in a database. The main aim of a DBMS is to supply a way to store up and retrieve database information that is both convenient and efficient. The DBMS is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system". Often the term "database" is also used to loosely refer to any of the DBMS, the database system or an application associated with the database.

MS Access is one of the most common examples of the DBMS software. Microsoft Access is a DBMS from Microsoft that combines the relational Microsoft Jet Database Engine with a GUI and software-development tools. It is a member of the MS Office suite of applications, included in the Professional and higher editions or sold separately. It stores data in its own format based on the Access Jet Database Engine. It can also import or link directly to data stored in other applications and databases.



**Fig. 10.1 Characteristics of Database and DBMS**

---

## 10.4 DBMS ARCHITECTURE

---

The architecture of the DBMS depends on the computer system on which it runs. For example, in a Client-Server DBMS architecture, the database systems at server machine can run several requests made by client machine. We will understand this communication with the help of diagrams. There are three types of DBMS architectures:

- Single tier architecture
- Two tier architecture

➤ Three tier architecture

---

### 10.4.1 SINGLE TIER ARCHITECTURE

---

In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database. For example, suppose you want to fetch the records of employee from the database and the database is available at your computer system. Therefore, the request to fetch employee details is generated by the computer and the records are fetched from the database by your computer as well. This type of system is generally referred as local database system.

---

### 10.4.2 TWO TIER ARCHITECTURE

---

In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in Fig. 10.2. Whenever client machine makes a request to access the database present at the server using a query language like SQL; the server performs the request on the database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for interaction between the server and client.

---

### 10.4.3 THREE-TIER ARCHITECTURE

---

In three-tier architecture, another layer is present between the client machine and server machine. In this architecture, the client application doesn't communicate directly with the database systems present at the server machine, rather the client application communicates with the server application, and the server application internally communicates with the database system present at the server as shown in Fig. 10.3.

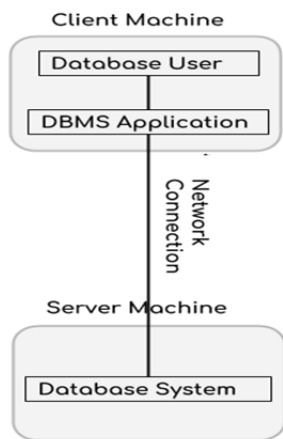


Fig. 10.2 Two-tier Architecture

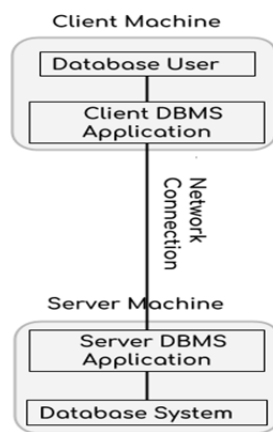


Fig. 10.3 Three-tier Architecture

## CHECK YOUR PROGRESS

- What do you mean by DBMS?
- What are the main characteristics of DBMS?
- Define the meaning of single tier architecture and two tier architecture in DBMS.

---

## 10.5 COMPONENTS OF DBMS

---

In order to facilitate these functions, DBMS has the following key components:

**Software-** DBMS is primarily a software system that can be considered as a management console or an interface to interact with and manage databases. The interfacing also spreads across real-world physical systems that contribute data to the backend databases. The OS, networking software, and the hardware infrastructure is involved in creating, accessing, managing, and processing the databases.

**Data-** DBMS contains operational data, access to database records and metadata as a resource to perform the necessary functionality. The data may include files with such as index files, administrative information, and data dictionaries used to represent data flows, ownership, structure, and relationships to other records or objects.

**Procedures-** While not a part of the DBMS software, procedures can be considered as instructions on using DBMS. The documented guidelines assist users in designing, modifying, managing, and processing databases.

**Database Languages-** These are components of the DBMS used to access, modify, store, and retrieve data items from databases; specify database schema; control user access; and perform other associated database management operations. Types of DBMS languages include Data Definition Language (DDL), Data Manipulation Language (DML), Database Access Language (DAL) and Data Control Language (DCL).

**Query Processor-** As a fundamental component of the DBMS, the query processor acts as an intermediary between users and the DBMS data engine in order to communicate query requests. When the users enter an instruction in SQL language, the command is executed from the high-level language instruction to a low-level language that the underlying machine can understand and process to perform the appropriate DBMS functionality. In addition to instruction parsing and translation, the query processor also optimizes queries to ensure fast processing and accurate results.

**Runtime Database Manager-** A centralized management component of DBMS that handles functionality associated with runtime data, which is commonly used for context-based database access. This component checks for user authorization to request the query; processes the approved queries; devises an optimal strategy



for query execution; supports concurrency so that multiple users can simultaneously work on same databases; and ensures integrity of data recorded into the databases.

**Database Manager-** Unlike the runtime database manager that handles queries and data at runtime, the database manager performs DBMS functionality associated with the data within databases. The database manager allows a set of commands to perform different DBMS operations that include creating, deleting, backup, restoring, cloning, and other database maintenance tasks. It may also be used to update the database with patches from vendors.

**Database Engine-** This is the core software component within the DBMS solution that performs the core functions associated with data storage and retrieval. A database engine is also accessible via APIs that allow users or apps to create, read, write, and delete records in databases.

**Reporting-** The report generator extracts useful information from DBMS files and displays it in structured format based on defined specifications. This information may be used for further analysis, decision making, or business intelligence.

---

## 10.6 NEED OF DBMS

---

A DBMS is a system software for easy, efficient and reliable data processing and management. It can be used for:

- Creation of a database
- Retrieval of information from the database
- Updating the database
- Managing a database

It provides us with the many functionalities and is more advantageous than the traditional file system in many ways listed below.

---

### 10.6.1 PROCESSING QUERIES AND OBJECT MANAGEMENT

---

In traditional file systems, we cannot store data in the form of objects. In practical-world applications, data is stored in objects and not files. So in a file system, some application software maps the data stored in files to objects so that can be used further.

We can directly store data in the form of objects in a DBMS. Application level code needs to be written to handle, store and scan through the data in a file system whereas a DBMS gives us the ability to query the database.

---

### 10.6.2 CONTROLLING REDUNDANCY AND INCONSISTENCY

---

Redundancy refers to repeated instances of the same data. A database system provides redundancy control whereas in a file system, same data may be stored multiple times. For example, if a student is studying two different

educational programs in the same college, say Engineering and History, then his information such as the phone number and address may be stored multiple times, once in Engineering department and the other in History department. Therefore, it increases time taken to access and store data. This may also lead to inconsistent data states in both places. A DBMS uses a very important feature i.e. data normalization to avoid redundancy and duplicates.

---

### **10.6.3 EFFICIENT MEMORY MANAGEMENT AND INDEXING**

---

DBMS makes complex memory management easy to handle. In file systems, files are indexed in place of objects so query operations require entire file scans whereas in a DBMS, object indexing takes place efficiently through database schema based on any attribute of the data or a data-property. This helps in fast retrieval of data based on the indexed attribute.

---

### **10.6.4 CONCURRENCY CONTROL AND TRANSACTION MANAGEMENT**

---

Several applications allow user to simultaneously access data. This may lead to inconsistency in data in case files are used. Consider two withdrawal transactions X and Y in which an amount of 100 and 200 is withdrawn from an account A initially containing 1000. Now since these transactions are taking place simultaneously, different transactions may update the account differently. X reads 1000, debits 100, updates the account A to 900, whereas Y also reads 1000, debits 200, updates A to 800. In both cases account A has wrong information. This results in data inconsistency. A DBMS provides mechanisms to deal with this kind of data inconsistency while allowing users to access data concurrently. A DBMS implements ACID (atomicity, durability, isolation, consistency) properties to ensure efficient transaction management without data corruption.

---

### **10.6.5 ACCESS CONTROL AND EASE IN ACCESSING DATA**

---

A DBMS can grant access to various users and determine which part and how much of the data can they access from the database thus removing redundancy. Otherwise in file system, separate files have to be created for each user containing the amount of data that they can access. Moreover, if a user has to extract specific data, then he needs a code/application to process that task in case of file system, e.g. Suppose a manager needs a list of all employees having salary greater than X. Then we need to write business logic for the same in case data is stored in files. In case of DBMS, it provides easy access of data through queries, (e.g., SELECT queries) and whole logic need not be rewritten. Users can specify exactly what they want to extract out of the data.

---

### **10.6.6 INTEGRITY CONSTRAINTS**

---

Data stored in databases must satisfy integrity constraints. For example,

Consider a database schema consisting of the various educational programs offered by a university such as (B.Tech/M.Tech/B.Sc/M.Sc/BCA/MCA) etc. Then we have a schema of students enrolled in these programs. A DBMS ensures that it is only out of one of the programs offered schema, that the student is enrolled in, i.e. Not anything out of the blue. Hence, database integrity is preserved.

---

## **10.7 ADVANTAGES OF DBMS**

---

The DBMS is preferred over the conventional file processing system due to the several benefits which are discussed below.

---

### **10.7.1 MINIMIZE DATA REDUNDANCY**

---

In File Processing System, duplicate data is created in many places because all the programs have their own files. This creates data redundancy which in turn wastes labor and space. In DBMS, all the files are integrated in a single database. The whole data is stored only once at a single place so there is no chance of duplicate data. For example- A student record in library or examination can contain duplicate values, but when they are converted into a single database, all the duplicate values are removed. Complete redundancy can be removed because somehow we need duplicate value to relate tables with each other. But still DBMS controls data redundancy that saves lots of labour and time.

---

### **10.7.2 SHARING OF DATA**

---

In DBMS, Data can be shared in between authorized user of database. All the users have their own right to access the database up to a level. Database Administration has complete access of database. He can assign users to access the database. Other users are also authorized to access database and also they can share data between them. Many users have same authority to access the database.

---

### **10.7.3 DATA CONSISTENCY**

---

The DBMS controls data redundancy which in turn controls data consistency. Data consistency means if you want to update data in any files then all the files should not be updated again. As in DBMS, data is stored in a single database so data becomes more consistent in comparison to file processing system. Also updated values are available to all the users immediately.

---

### **10.7.4 DATA INTEGRITY**

---

Data integrity means unification of so many files into a single file. In DBMS data is stored in different tables. A database contains different tables that are linked to each other. Many users feed entries in these tables so it is important to maintain data items and association between data items. DBMS allows data integrity that makes it easy to decrease data duplicity. Data integration reduces redundancy as well as data inconsistency.

---

## **10.7.5 SEARCH CAPABILITY**

---

Users of database may require to fetch data from the database. There are numerous queries users may ask about the data. Search speed of the database must be fast to produce quick results. If users execute any query, then it is required that he get fastest results from the database. It is an objective of database to maintain flexible search capability.

---

## **10.7.6 SECURITY**

---

Data security means protecting your precious data from unauthorized access. Data in database should be kept secure and safe to unauthorized modifications. Only authorized users should have the grant to access the database. There is a username set for all the users who access the database with password so that no other person could access this information. The DBMS always keep database tamperproof, secure and theft free.

---

## **10.7.7 PRIVACY**

---

Privacy, in the broadest sense, is the right of individuals, groups, or organizations to control who can access, observe, or use something they own, such as their bodies, property, ideas, data, or information. Control is established through physical, social, or informational boundaries that help prevent unwanted access, observation, or use. Privacy is the very important feature and it is handled in DBMS very carefully and intelligently. So many examples are there where privacy is experienced. For example, many social networking platforms are there where others in one's friend have access to one's content upto some extent. Or in other words, privacy means up to what extent a user can access the data. It is predetermined by the DBA that who will access the data and up to what level he will be able to access it. Let say when you make a Facebook page then you have the power to give rights to other users that who will be the promoter, editor and admin.

---

## **10.7.8 SIMPLICITY**

---

It is easy to implement all levels of DBMS like we normally know in normalization. Normalization is a way by which the all type of redundant data insertion can be avoided and in a very simple but powerful way. DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database. Simplicity means to represent the overall logical view of data in a simple and clear manner. The DBMS is very simple for its users who use it. All the operations like create, insert, delete, alter, and update are very easy to implement.

---

## **10.7.9 BACKUP AND RECOVERY**

---

Data loss is a very big problem for all the organizations. In traditional file processing system, a user needs to backup the database after a regular interval of

time that wastes lots of time and resources. If the volume of data is large then this process may take a very long time. DBMS solves this problem of taking backup again and again because it allows automatic backup and recovery of database. For examples, if a system fails in the middle of any process then DBMS stores the values of that state in which database were before query execution.

---

### **10.7.10 INTEGRITY CONSTRAINTS**

---

Constraints are used to store accurate data because there are many users who feed data in database. Data stored in database should always be correct and accurate. DBMS provides the capability to enforce these constraints on database. For example, the maximum marks obtained by the students can never be more than 100. Also account balance of Banks like Axis should not be less than 2500 otherwise you will be penalized.

---

### **10.7.11 DATA ATOMICITY**

---

Any complete transaction in database is called atomic unit. It is the duty of DBMS to store a complete transaction in database. If any transaction is partially completed, then it rolls back them. For example, in railway reservation system, if user has completed the process of ticket reservation then his record will be stored; and amount of money will be deducted from his account otherwise no amount will be deducted and if deducted it will be given back.

---

### **10.7.12 DEVELOPMENT OF NEW APPLICATIONS**

---

For IT, providing the technology to support new applications without adding additional complexity and risk, and while staying within budget, are the big challenges. IT must sort through a maze of technical options, beginning with the right database management system (DBMS). If a new application is required and data is available for creating the application, then it is very easy to develop new application. No time will be consumed in creating stored data again and again. As the data is already available and which can be shared easily.

---

### **10.7.13 CONCURRENCY CONTROL**

---

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. If two users are accessing data simultaneously and they both want to update values of same records, then it may create the problem of concurrency and it should be controlled. The DBMS has the power to control concurrency so that no transactions are lost.

---

### **10.7.14 DATA MIGRATION**

---

Database migration means moving your data from one platform to another. There are many reasons you might want to move to a different platform.

For example, a company might decide to save money by moving to a cloud-based database. Or, a company might find that some particular database software has features that are critical for their business needs. Or, the legacy systems are simply outdated. The process of database migration can involve multiple phases and iterations — including assessing the current databases and future needs of the company, migrating the schema, and normalizing and moving the data. Data migration means adjusting storage of data according to its popularity. In a database, there is some kind of data that is accessed frequently and at the same time some data is accessed occasionally. So it is required to store frequently accessed data in a manner that it can be accessed quickly.

---

### **10.7.15 TUNABILITY**

---

Database tuning describes a group of activities used to optimize and homogenize the performance of a database. It usually overlaps with query tuning, but refers to design of the database files, selection of the database management system (DBMS) application, and configuration of the database's environment (operating system, CPU, etc.). Tuning means adjusting something to get better performance. Same in the case of DBMS, as it provides tunability to improve performance. DBA adjust database to get effective results.

---

### **10.7.16 SOLVES ENTERPRISE AND INDIVIDUAL REQUIREMENT**

---

A DBMS provides a wide range of user interfaces to use a database. There are many users working on the database having a different level of knowledge. So it is desirable that the DBMS gives GUI to all the users. Still Enterprise requirement is more than any users so DBMS focus mainly on the Enterprise requirement.

---

## **10.8 DISADVANTAGES OF DBMS**

---

Though DBMS has many benefits however it has some disadvantages too which are explained below.

---

### **10.8.1 INCREASED COST**

---

**Cost of Hardware and Software**— This is the first disadvantage of database management system. This is because, for DBMS it is mandatory to have a high speed processor and also a large memory size because nowadays there is a large amount of data in every field which needs to be stored safely and with a security. The requirement of these large amount of space and a high speed processor needs an expensive hardware and also an expensive software too. That is there is a requirement of sophisticated hardware and software which means that we need to upgrade the hardware which is used for file-based system. Hardware and Software, both requires maintenance which costs very high. All the operating, Training (all levels including programming, application development, and database administration), licensing, and regulation compliance costs very high.

**Cost of Staff Training**– Educated staff (database administrator, application programmers, data entry operations) who maintains the DBMS also requires good amount of salary. We need the database system designers to be hired along with application programmers. Alternatively, the services of some software house need to be taken. So there is a lot of money which needs to be spent for developing software.

**Cost of Data Conversion**– We need to convert our data into database management system, there is a requirement of lot of money as it adds on to the cost of the DBMS. This is because, for this conversion we need to hire database system designers whom we have to pay a lot of money and also services of some software house will be required. All this shows that a high initial investment for hardware, software and trained staff is required by DBMS. So, altogether DBMS results in a costlier system.

---

## **10.8.2 COMPLEXITY**

---

As we all know that nowadays all companies are using the DBMS as it fulfils lots of requirement and also solves the problem. But a problem arises, that is all these functionality has made it an extremely complex software. For the proper requirement of DBMS, it is very important to have a good knowledge of it by the developers, DBA, designers and also the end users. This is because if any one of them do not acquire a proper and complete skills than this may lead to data loss or database failure.

These failures may lead to bad design decisions due to which there may be a serious and bad consequences for the organization. So this complex system needs to be understood by everyone using it. As it cannot be managed very easily. All this shows that the DBMS is not a child's game as it cannot be managed very easily. It requires a lot of management. A good staff is also needed to manage the database at the times when it becomes very complicated to decide where to pick data from and where to save it.

---

## **10.8.3 CURRENCY MAINTENANCE**

---

This is necessary to keep the system updated because efficiency, which is one of the biggest factor and need to be overlook must be maximized. That is, we need to maximize the efficiency of the database system to keep our system advance. For this, frequent updating must be performed on all the components as new threats enter daily. The DBMS should be updated according to the current scenario. Also security measures must be needed. Due to advancement in database technology, training cost tends to be significant.

---

## **10.8.4 PERFORMANCE**

---

Traditional file system is written for small organizations and for some specific applications due to which performance is generally very good. But for the small scale firms, the DBMS does not give a good performance as its speed is very slow. As a result, some applications would not run as fast as they could. Hence it is not good to use DBMS for the small firms. Because performance is a

factor which is overlooked by everyone. If performance is good than everyone (developers, designers, end users) would use it easily and it would be user friendly too as speed of the system totally depends on the performance and performance needs to be good.

---

### **10.8.5 FREQUENCY UPGRADE/REPLACEMENT CYCLES**

---

Nowadays we need to stay up-to-date about the latest technologies as new developments are arriving in the market frequently. Frequent upgrade of the products is done by the DBMS vendors in order to add new functionality to the systems. New upgrade versions of the software often come bundled. Sometimes these updates also need hardware upgrades. Sometimes these changes and updating are so fast that the users feel it difficult to work with that system because it is uneasy to learn new commands and understanding them again and again when the new upgrades are done. All these upgrades also costs money in order to train users, designers, etc. to use the new features.

#### **CHECK YOUR PROGRESS**

- Write the main components of DBMS.
- Give any three advatages of DBMS.
- Discuss about any three disadvantages of DBMS.

---

## **10.9 DATABASE ADMINISTRATOR**

---

A Database Administrator or DBA in short, is a person or a group of person who are responsible for managing all the activities related to database system. This job requires a high level of expertise by a person or group of persons. There are very rare chances that only a single person can manage all the database system activities so companies always have a group of people who take care of database system. In a nut shell, the DBA is the controller of everything related to database system.

The DBA is responsible for installing the database software. He configures the software of database and then upgrades it if needed. There are database software like Oracle, Microsoft SQL, and MySQL in the industry, so DBA decides installation and configuration of these database software will take place. The major role duties of a DBA are discussed as follows.

---

### **10.9.1 DECIDING THE HARDWARE DEVICE**

---

Depending upon the cost, performance and efficiency of the hardware, it is the DBA who has the duty of deciding which hardware device would suit the company requirement. It is hardware that is an interface between end users and database so it must be of the best quality. Moreover, where is the system going to be located (physically)? Will you take the system in-house or engage the services of a company to host the data and the software system for you? This could have implications for support, cost (including any additional hardware you would require), security, and possibly speed.



---

## **10.9.2 MANAGING DATA INTEGRITY**

---

Data should be pure and complete; it is known as data integrity. It is of two types- 1) Entity integrity and 2) Referential integrity. Entity integrity is basically intended with the primary used in the table. Referential integrity on the other hand deals with the foreign key and related with the primary on some table. Data integrity should be managed accurately because it protects the data from unauthorized access. The DBA manages relationship between the data to maintain data consistency.

---

## **10.9.3 DECIDING DATA RECOVERY AND BACK UP METHOD**

---

DBAs are responsible for making a comprehensive backup plan for databases for which they are accountable. If any company is having a big database, then it is likely to happen that database may fail at any instance. It is required that a DBA takes backup of entire database in regular time span. DBA has to decide that how much data should be backed up and how frequently the backup should be taken. Also the recovery of database is done by the DBA. It is imperative that the DBA be aware of database and related OS and application components that need to be backed up, whether via an online backup or an offline cold backup.

---

## **10.9.4 TUNING DATABASE PERFORMANCE**

---

Database performance plays an important role for any business. If user is unable to fetch data speedily then it may loss company business. So by tuning and modifying SQL commands a DBA can improve the performance of database. Configuration of the database for optimal performance can be done by following the tuning guidelines like- 1) Good database design 2) Disk I/O optimization 3) Checkpointing 4) Disk and database overhead 5) Increasing concurrency etc.

---

## **10.9.5 CAPACITY ISSUES**

---

All the databases have their limits of storing data in it and the physical memory too have some limitations. The DBA has to decide the limit and capacity of database and all the issues related to it. Although storing data on file systems is intuitively easier to understand, and all databases must ultimately store data on the file system, there are problems with storing data in files and on file systems: 1) Access times and overhead 2) Data aggregation 3) Concurrency.

There's also a "catch-22" situation here: if you have lots of data in a single file, you run into problems where reading data takes too long. If you put little bits of data into lots of different files you can typically skirt around the concurrency issues and to a lesser extent the data aggregation issues, but you spend more time finding files, and if the "little bits" of data is consistently below (or around) the block size (i.e. 4kb) of your file system, the file system can't store data efficiently.

---

## **10.9.6 DATABASE DESIGN**

---

Database design is the process to define and represent entities and relations. A structured approach that uses procedures, techniques, tools, and documentation help to support and make possible the process of design is called Design Methodology. A design methodology encapsulates various phases, each containing some stages, which guide the designer in the techniques suitable at each stage of the project. A design methodology also helps the designer to plan, manage, control, and evaluate database development and managing projects. Logical design of the database is designed by the DBA. Also a DBA is responsible for physical design, external model design, and integrity control. The main database design principles are based on the normalization, and intended to represent each entity as a table, selecting the primary key, assigning entity attributes to fields etc.

---

## **10.9.7 DATABASE ACCESSIBILITY**

---

The DBA writes subschema to decide the accessibility of database. He decides the users of the database and also which data is to be used by which user. No user has the power to access the entire database without the permission of DBA. The parts of access database are- Tables, Forms, Reports, Queries, Macros, Modules. Using Access, you can:

- Add new data to a database, such as a new item in an inventory
- Edit existing data in the database, such as changing the current location of an item
- Delete information, perhaps if an item is sold or discarded
- Organize and view the data in different ways
- Share the data with others via reports, e-mail messages, an intranet, or the Internet

---

## **10.9.8 DECIDES VALIDATION CHECKS ON DATA**

---

The DBA has to decide which data should be used and what kind of the data is accurate for the company. So he always puts validation checks on data to make it more accurate and consistence. Validation is a process whereby the data entered in the database is checked to make sure that it is sensible. For example, validation can be utilized to check that only Male or Female is entered in a sex field. It cannot check whether or not the data entered is correct. It can only check whether or not the data makes sense. Validation is a way of trying to lessen the number of errors during the process of data input. Validation is carried out by the computer when you input data. It is a way of checking the input data against the set of validation rules. The purpose of validation is to make sure that data is a) logical, b) rational, and c) complete and within acceptable limits.

---

## 10.9.9 MONITORING PERFORMANCE

---

If database is working properly then it doesn't mean that there is no task for the DBA. Yes of course, he has to monitor the performance of the database. A DBA monitors the CPU and memory usage. Database monitoring is the tracking of database performance and resources in order to create and maintain a high performance and highly available application infrastructure. For example, categories for SQL Server, MySQL and Oracle database monitoring include:

- Query details (top CPU, slow running, and most frequent)
- Session details (current user connections and locks)
- Scheduled jobs
- Replication details
- Database performance (buffer, cache, connection, lock, and latch)

---

## 10.9.10 DECIDES CONTENT OF THE DATABASE

---

A database system has many kind of content information in it. DBA decides fields, types of fields, and range of values of the content in the database system. One can say that the DBA decides the structure of database files. Understanding the purpose of your database will inform your choices throughout the design process. Make sure you consider the database from every perspective. For instance, if you were making a database for a public library, you'd want to consider the ways in which both patrons and librarians would need to access the data. Here are some ways to gather information before creating the database:

- Interview the people who will use it
- Analyze business forms, such as invoices, timesheets, surveys
- Comb through any existing data systems (including physical and digital files)

---

## 10.9.11 PROVIDES HELP AND SUPPORT TO USER

---

If any user needs help at any time, then it is the duty of the DBA to help him. Complete support is given to the users who are new to database by the DBA. End users are basically those people whose jobs require access to the database for querying, updating and generating reports. The database primarily exists for their use. There are several categories of end users these are as follows:

1. **Casual End Users**– These are the users who occasionally access the database but they require different information each time. They use a sophisticated database query language basically to specify their request and are typically middle or level managers or other occasional browsers. These users learn very few facilities that they may use repeatedly from the multiple facilities provided by DBMS to access it.
2. **Naive or parametric end users**– These are the users who basically make

up a sizeable portion of database end users. The main job function revolves basically around constantly querying and updating the database for this we basically use a standard type of query known as canned transaction that have been programmed and tested. These users need to learn very little about the facilities provided by the DBMS they basically have to understand the users' interfaces of the standard transaction designed and implemented for their use.

3. **Sophisticated end users**– These users basically include engineers, scientist, business analytics and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their application to meet their complex requirement. These users try to learn most of the DBMS facilities in order to achieve their complex requirements.
4. **Standalone users**– These are those users whose job is basically to maintain personal databases by using a ready-made program package that provides easy to use menu-based or graphics-based interfaces, an example is the user of a tax package that basically stores a variety of personal financial data of tax purposes. These users become very proficient in using a specific software package.

---

### 10.9.12 DATABASE IMPLEMENTATION

---

Database must be implemented before anyone can start using it, so the DBA implements the database system. The DBA has to supervise the database loading at the time of its implementation. The implementation phase is where you install the DBMS on the required hardware, optimize the database to run best on that hardware and software platform, and create the database and load the data. The initial data could be either new data captured directly or existing data imported from a MariaDB database or another DBMS. You also establish database security in this phase and give the various users that you've identified access applicable to their requirements. Finally, you also initiate backup plans in this phase. The following are steps in the implementation phase:

- Install the DBMS.
- Tune the setup variables according to the hardware, software and usage conditions.
- Create the database and tables.
- Load the data.
- Set up the users and security.
- Implement the backup regime.

---

### 10.9.13 IMPROVE QUERY PROCESSING PERFORMANCE

---

Queries made by the users should be performed speedily. As we have discussed that users need fast retrieval of answers so the DBA improves query

processing by improving their performance. Improving Query Processing Performance in Database Management Systems has been a research challenge. This is the most important and is a real problem, this happens to be very crucial in large organizations with heterogeneous data, online system, billing systems and so on. The Performance monitoring has been evaluated and used by various tools. Most DBA's agreed that these tools are valuable.

---

## **10.10 OPEN DATABASE CONNECTIVITY**

---

An ODBC driver uses the Open Database Connectivity (ODBC) interface by Microsoft that allows applications to access data in DBMS using SQL as a standard for accessing the data. An ODBC driver uses the ODBC interface by Microsoft that allows applications to access data in DBMS using SQL as a standard for accessing the data. ODBC permits maximum interoperability, which means a single application can access different DBMS. Application end users can then add ODBC database drivers to link the application to their choice of DBMS.

---

### **10.10.1 INTRODUCTION TO ODBC**

---

ODBC, is an API that lets software connect with the DBMS while remaining independent of them. This is important, because it allows applications to interact with multiple databases simultaneously using SQL. For organizations that have multiple data streams and must store them on separate databases, ODBC offers a solution that lets them use the software they need without having to worry about which DBMS they have to use. It's useful to think of ODBC as a peripheral driver which lets specific tools connect to a program. Much like printers require the specific instructions to allow them to connect with multiple different computers and devices, ODBC is a bridge between applications and the databases they require.

Additionally, ODBC allows organizations to centralize their data streams into a single application or dashboard more efficiently. ODBC works by creating a link between the application and the database, taking queries from end users and translating it for the DBMS to process them. Developers connect ODBC's API tools to the DBMS by using specific drivers. Applications then call specific functions installed in these drivers to access the data they need from specific sources.

---

### **10.10.2 ODBC HISTORY**

---

Microsoft introduced the ODBC standard in 1992. It was a standard designed to unify access to SQL databases. Following the success of ODBC, Microsoft introduced OLE DB which was to be a broader data access standard. OLE DB was a data access standard that went beyond just SQL databases and extended to any data source that could deliver data in tabular format. Microsoft's plan was that OLE DB would supplant ODBC as the most common data access standard.

More recently, Microsoft introduced the ADO data access standard. The ADO was supposed to go further than OLE DB, in that ADO was more object

oriented. However, even with Microsoft's very significant attempts to replace the ODBC standard with what were felt to be "better" alternatives, the ODBC has continued to be the defacto data access standard for SQL data sources. In fact, today the ODBC standard is more common than OLE DB and ADO because ODBC is widely supported (including support from Oracle and IBM) and is a cross platform data access standard. Today, the most common data access standards for SQL data sources continue to be ODBC and JDBC, and it is very likely that standards like OLE DB and ADO will fade away over time.

---

### 10.10.3 ODBC ARCHITECTURE

---

The ODBC architecture consists of the components given below.

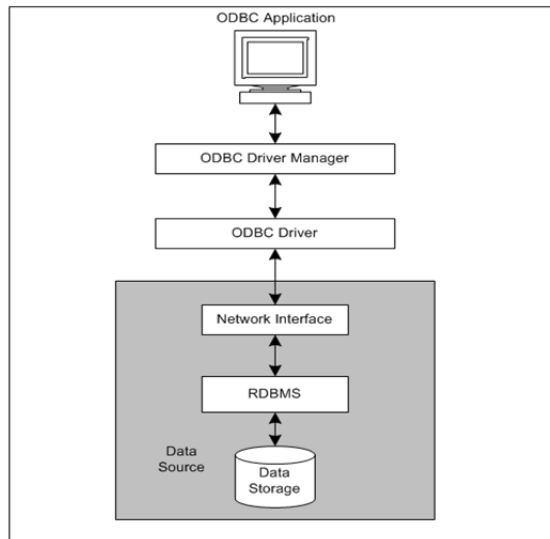
**Application**— An ODBC application is any program that calls ODBC functions and uses them to issue SQL statements.

**ODBC Driver Manager**— The driver manager routes call from an application to the ODBC driver. The ODBC driver manager loads the requested driver in response to an application's call to the ODBC SQLConnect or SQLDriverConnect functions.

**ODBC Driver**— An ODBC driver is a dynamic link library (DLL) or a shared library that processes ODBC function calls for a specific data source. The driver connects to the data source, translates the standard SQL statements into syntax the data source can process, and returns data to the application.

**Data source** — A data source is a combination of a database system, the operating system it uses, and any network software required to access it.

Fig. 10.4 shows the components of an ODBC environment.



**Fig. 10.4 Components of an ODBC environment**

(Source:[https://documentation.progress.com/output/ua/OpenEdge\\_latest/index.html#page/dmsdv/odbc-architecture.html](https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/dmsdv/odbc-architecture.html))

---

## 10.10.4 USE OF ODBC

---

For organizations that use multiple DBMS and streams, ODBC is one of the easiest ways to centralize and manage data without having to use multiple systems at the same time. One of the clearest use cases for ODBC is for creating dashboards. For most organizations, dashboards—even specific ones—tend to draw data from multiple internal and sometimes external sources. As such, using an ODBC connector can improve several areas of the analytics process. The ODBC lets developers connect their existing data visualization tools to any database, greatly increasing their accuracy and depth. Companies that constantly interact with multiple databases simultaneously for analytics can also optimize their querying ability and draw information from a broader range of sources, as well as create more granular reports.

Additionally, the ODBC allows companies to sort and store their data more efficiently. Instead of forcing a single DBMS to handle data, it may not be optimally suited for, organizations, can instead mash up multiple sources without worrying about their compatibility or accessibility.

---

## 10.11 DATABASE ACCESS

---

When it comes to implementing a data access solution in Visual Basic applications, you currently have three choices- Data Access Objects (DAO), Remote Data Objects (RDO), and ActiveX Data Objects (ADO). In these three, it would be examined each option, noting their similarities and differences. It would also be looked at some cases where one is better suited for a specific task than another.

---

### 10.11.1 USING DAO FOR DATA ACCESS

---

DAO is an API available with Microsoft's Visual Basic that lets a programmer request access to a Microsoft Access database. DAO was Microsoft's first object-oriented interface with databases. DAO objects encapsulate Access's Jet functions. Through Jet functions, it can also access other SQL databases.

---

#### 10.11.1.1 DAO BASICS

---

DAO, which was created before RDO and ADO, is a set of objects that enables client applications to programmatically access data. But DAO doesn't just let you access data—it also lets you control and manage local and remote databases in various formats. Using DAO, you can create and modify the database structure; create tables, queries, relationships, and indexes; retrieve, add, update, and remove data; implement security; work with different file formats; and link tables to other tables.

---

#### 10.11.1.2 DAO OBJECTS

---

To understand DAO better, let's look at the DAO objects given in Table 10.1

**Table 10.1. List of Common DAO Objects**

(Source: <https://www.techrepublic.com/article/using-dao-for-data-access-in-your-vb-apps/>)

<b>Object</b>	<b>Description</b>
DBEngine	The top-level object in the DAO object hierarchy
Workspace	An active DAO session
Connection	Network connection to an ODBC database
Database	Open database
Error	Data access error information storage
Field	A field in a <i>TableDef</i> , <i>QueryDef</i> , <i>Recordset</i> , <i>Index</i> , or <i>Relation</i> object
QueryDef	Saved query definition in a database
Recordset	Set of records defined by a table or query
TableDef	Saved table definition in a database
User	User account in the current workgroup
Index	Table index
Parameter	Query parameter
Property	Property of an object

---

### 10.11.1.3 DAO WITH JET

---

Microsoft Jet objects include TableDef, QueryDef, Field, Index, Parameter, Relation, Recordset, User, Group, Container, and Document. The DBEngine object contains two collections- Workspaces and Errors. The Workspaces collection is the default collection of the DBEngine, so you don't have to refer to it explicitly. When you don't specifically create a new Workspace object, DAO will create one for you. The setting of the DefaultType property of DBEngine determines what type of workspace is created for Microsoft Jet or ODBCdirect. The default value of this property is dbUseJet, but you can explicitly set it to dbUseODBC as the type argument of the CreateWorkspace method.

The Workspace object defines a session for a user based on users' permissions and allows managing of the current session. It also contains open databases and offers mechanisms for simultaneous transactions and for securing the application. The Fields collection is the default collection for TableDef, QueryDef, Index, Relation, and Recordset objects. Recordset objects can be of the three types- Table, Dynaset, Snapshot, or Forward-Only.

---

### 10.11.1.4 DAO WITH ODBC DIRECT

---

The DAO ODBCdirect object model includes a subset of the objects in a Microsoft Jet workspace and the Connection object. To establish a connection using ODBCdirect, you have to use the OpenConnection method on a new Connection object or the OpenDatabase method to open a new Database object. A Connection object represents a connection to an ODBC database in an ODBCdirect workspace. The Connections collection contains all currently open Connection objects. When you open a Connection object, it is automatically appended to the Connections collection of the Workspace object. When you close a Connection object with the Close method, the Connections object is removed



from the Connections collection. In addition to the Table, Dynaset, Snapshot, and Forward-Only types of Recordsets, ODBCDirect offers the Dynamic type.

---

### **10.11.1.5 ADVANTAGES AND DISADVANTAGES OF USING DAO**

---

On the plus side, DAO is fairly easy to use. And since DAO has been around longer than RDO or ADO and has been used in more projects, it pays to know how DAO works. Furthermore, if your application is running in a 16-bit environment, DAO is your only choice.

But DAO is older technology, and it doesn't offer as much functionality as RDO and ADO. For instance, ADO can provide an interface to e-mail and file systems and custom business objects, as well as other sources. Microsoft is now focusing most of its improvements and advances on ADO, as well.

Generally, it's better to use DAO for accessing local databases where the speed is not the top priority and the number of users is limited, and to use either RDO or ADO for accessing remote databases and for larger scale projects.

---

### **10.11.2 USING RDO FOR DATA ACCESS**

---

Remote Data Objects (RDO) is specifically designed to access remote ODBC relational data sources, and makes it easier to use ODBC without complex application code. The RDO is a primary means of accessing SQL Server, Oracle, or any relational database that is exposed with an ODBC driver. It is an API from Microsoft that lets programmers writing Windows applications get access to and from both Microsoft and other Database Providers. In turn, RDO statements in a program use Microsoft's lower-layer Data Access Objects (DAO) for actual access to the database. Database providers write to the DAO interface. RDO has evolved into ActiveX Data Objects (ADO) which is now the program interface Microsoft recommends for new programs. ADO also provides access to nonrelational databases and is somewhat easier to use.

Moreover, the RDOs are used to access a remote database through ODBC. Accessing an ODBC data source using RDO is faster than accessing the same using DAO. Remote Data Control is an ActiveX control that is used to access an ODBC data source using RDOs. This is same as a Data Control, except that the data control uses DAOs and Remote data control uses RDOs.

The general characteristics of RDO are:

- Simplicity (when compared to the ODBC API).
- High performance against remote ODBC data sources.
- Programmatic control of cursors.
- Complex cursors, including batch.
- Ability to return multiple result sets from a single query.
- Synchronous, asynchronous, or event-driven query execution.
- Reusable, property-changeable objects.

- Ability to expose underlying ODBC handles (for those ODBC functions that are not handled by RDO).
- Excellent error trapping.

First let us understand RDO object model and key features of RDO and ODBC. Then we will understand how to use Remote Data Control and RDOs to access Oracle through ODBC. RDO object model is very small compared with DAO object model. Here is the list of objects in RDO object model. And most of them have their counterparts in DAO. The brief description about each object is given in Table 10.2.

**Table 10.2 Objects in RDO Object Model**

RDO Object	Description
rdoEngine	The base object. Created automatically when you first access RDO in your application.
rdoError	Used to handle all ODBC errors and messages generated by RDO. Created automatically.
rdoEnvironment	Defines a logical set of connections and transaction scope for a particular user name. Contains both open and allocated (but unopened) connections, provides mechanisms for simultaneous transactions, and provides a security context for data manipulation language (DML) operations on the database. rdoEnvironments(0) created automatically.
rdoConnection	Represents an open connection to a remote data source and a specific database on that data source, or an allocated but as yet unconnected object, which can be used to subsequently establish a connection.
rdoTable	Represents the stored definition of a base table or an SQL view.
rdoResultset	Represents the rows that result from running a query.
rdoColumn	Represents a column of data with a common data type and a common set of properties.
rdoQuery	An SQL query definition that can include zero or more parameters.
rdoParameter	Represents a parameter associated with an rdoQuery object. Query parameters can be input, output, or both.

---

### 10.11.3 RDO VS. DAO

---

Fundamentally RDO is same as DAO; and Remote data control is same as Data control. However, the RDO was designed and implemented strictly for relational databases that are accessed using ODBC. The RDO doesn't have a query engine like DAO, instead it depends on the query processor of the database that it is accessing.

Moreover, compared to the older Data Access Objects (DAO) technology, RDO is a smaller, faster, more sophisticated alternative. RDO is especially capable of building and executing queries against stored procedures and handling all types of result sets, including those generated by multiple result set procedures, those returning output arguments and return status, and those requiring complex input parameters.

---

### 10.11.4 USING ADO FOR DATA ACCESS

---

ActiveX Data Objects (ADO) is designed to be an easy-to-use application-level interface to any OLE DB data provider, including relational and non-relational databases, e-mail and file systems, text and graphics, and custom business objects, as well as existing ODBC data sources. Virtually all of the data available throughout the enterprise is available using the ADO data access technology. It is easy to use, language-independent, implemented with a small footprint, uses minimal network traffic, and has few layers between the client application and the data source— all to provide lightweight, high-performance data access.

The general characteristics of ADO are:

- Ease of use.
- High performance.
- Programmatic control of cursors.
- Complex cursor types, including batch and server and client-side cursors.
- Ability to return multiple result sets from a single query.
- Synchronous, asynchronous, or event-driven query execution.
- Reusable, property-changeable objects.
- Advanced recordset cache management.
- Flexibility — it works with existing database technologies and all OLE DB providers.
- Excellent error trapping.

## CHECK YOUR PROGRESS

- Who is DBA? Give any two responsibility of DBA.
- Why ODBC is required?
- Describe any two advantages of DAO.

---

## 10.12 STRUCTURED QUERY LANGUAGE

---

Structured Query Language (SQL), pronounced as "sequel", is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables. SQL offers two main advantages over older read–write APIs such as ISAM or VSAM. Firstly, it introduced the concept of accessing many records with one single command. Secondly, it eliminates the need to specify how to reach a record, e.g. with or without an index.

Originally based upon relational algebra and tuple relational calculus, the SQL consists of many types of statements, which may be informally classed as sublanguages, commonly: a data query language (DQL), a data definition language (DDL), a data control language (DCL), and a data manipulation language (DML). The scope of SQL includes data query, data manipulation (insert, update and delete), data definition (schema creation and modification), and data access control. Although SQL is essentially a declarative language (4GL), it includes procedural elements too.

SQL was one of the first commercial languages to utilize Edgar F. Codd's relational model. The model was described in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks." Despite not entirely adhering to the relational model as described by Codd, it became the most widely used database language. Moreover, the SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, most SQL code is not completely portable among different database systems without adjustments.

---

### 10.12.1 CREATE TABLE STATEMENT

---

The CREATE TABLE statement is used to create a new table in a database. In that table, if you want to add multiple columns, use the syntax below.

**Syntax:**

```
CREATE TABLE table_name (column1 datatype,  
column2 datatype,...);
```

The column parameters specify the names of the columns of the table. The data type parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

**Example:**

```
CREATE TABLE Employee( EmpId int, LastName varchar(255),  
  FirstName varchar(255), Address varchar(255),  
  City varchar(255) );
```

The EmpId column is of type int and will hold an integer. The LastName, FirstName, Address, and City columns are of type varchar and will hold characters and the maximum length for these fields is 255 characters.

---

### 10.12.2 INSERTING VALUES

---

The INSERT INTO statement is used to insert new records in a table. It is possible to write the INSERT INTO statement in two ways.

**Syntax:**

*First way:* It specifies both the column names and the values to be inserted. If you are adding values for all the columns of the table, then no need to specify the column names in the SQL query. However, make sure that the order of the values is in the same order as the columns in the table.

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

*Second way:*

```
INSERT INTO table_name  
VALUES (value1, value2, value3, ...);
```

**Example:** *Insert value in a First way.* The column names are used here

```
INSERT INTO Employee(EmpId,LastName,FirstName,ADDRESS,City)  
VALUES (1, 'XYZ', 'ABC', 'India', 'Mumbai' );  
  
INSERT INTO Employee (EmpId,LastName,FirstName,ADDRESS,City)  
VALUES (2, 'X', 'A', 'India', 'Pune' );
```

Insert value in *Second way*

```
INSERT INTO Employee  
VALUES (3, 'XYZ', 'ABC', 'India', 'Mumbai' );
```

---

### 10.12.3 SELECT STATEMENT IN SQL

---

The SELECT statement is used to select data from a database. The data returned

is stored in a result table, called the result-set.

```
SELECT column1, column2, ...  
FROM table_name;
```

Here, column1, column2, ... are the field names of the table you want to select from the data. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

If the above query is executed, then all record is displayed.

**Example:**

```
Select EmpId, LastName from Employee;  
Select * from Employee;
```

---

### 10.12.4 UPDATE STATEMENT IN SQL

---

The UPDATE statement is used to modify the existing records in a table.

**Syntax:**

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

**Example:**

```
UPDATE Employee  
SET FirstName= 'Krishan', City= 'Haridwar'  
WHERE EmpId= 1;
```

If the above query is executed then for EmpId= 1, "Firstname" and "City" column data will be updated.

---

### 10.12.5 DELETE STATEMENT IN SQL

---

The DELETE statement is used to delete existing records in a table for a particular Record.

**Syntax:**

```
DELETE FROM table_name WHERE condition;  
DELETE FROM Employee WHERE EmpId=1;
```

In Employee table EmpId = 1 record gets deleted.

---

## 10.13 DATABASE ACCESS WITH THE DATA CONTROL

---

The Data control does a lot "behind the scenes" and you may be tempted to use it in your applications, but be advised that the Data control is rarely used in professional applications – the norm is to write your own database access code so that you have complete control over the process. The intrinsic Data control is geared toward MS-Access 97 and earlier databases, although a later VB service pack added connectivity for Access 2000 databases. These articles use the two sample Access databases provided with Visual Basic (BIBLIO.MDB and NWIND.MDB). These databases are provided in Access 97 format. On a default installation of VB6, these databases can be found in the folder: C:\Program Files\Microsoft Visual Studio\VB98.

To do the exercises, you should make a folder into which you would copy the two database files mentioned above. Then, within the folder, make separate subfolder for each exercise, one level below the root of your folder. The DatabaseName property for the Data control in these exercises assumes that the database file resides one directory level above the folder in which the exercise project files reside.

---

### 10.13.1 CONNECTING TO AN ACCESS DATABASE USING THE VB DATA CONTROL

---

Normally 5-6 steps are used to access database using data control, given below:

**Step 1:** Open a new Visual Basic project.

**Step 2:** Put a data control (an intrinsic control, located in the VB toolbox) on the form and set the properties shown in Table 10.3.

**Table 10.3 Properties of the data control**

Property	Value
(Name)	datAuthors
Caption	Use the arrows to view the data
Connect	Access (default)
DatabaseName	..\biblio.mdb
DefaultType	UseJet (default)
RecordSource	Authors (choose from list)

**Note:-** When you use the Data Control in a project, the properties that must be set are DatabaseName and RecordSource, in that order. DatabaseName is the name of the database you want to use, and the RecordSource is the name of the table in that database that you want to use.

**Step3:** On your form, create a text box for each field in the Authors table, with

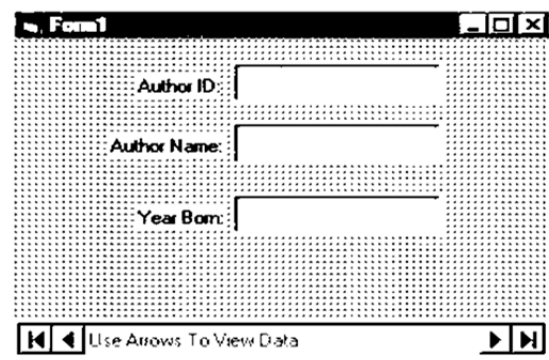
labels. (If you were to open the database in Access, you would see that the three fields of the Authors table are Au\_ID, Author, and Year Born.) Set the properties of the three textboxes as given in Table 10.4.

**Table 10.4 Properties of text boxes**

Name	DataSource	DataField
txtAuthID	datAuthors	Au_ID
txtAuthor	datAuthors	Author
txtYearBorn	datAuthors	Year Born

In addition, set the Enabled property of txtAuthID to False. When you want a control (such as a text box) to display data from a database, the properties that must be set are DataSource and Datafield. The DataSource is the name of the data control on the form (it should already be configured), and the DataField is the name of the particular field in the database that should be displayed in the control (this field will be in the table that was chosen for the RecordSource of the data control).

**Step4:** At this point, your form should resemble the screen-shot below (Fig. 10.5).



**Fig. 10.5 Data control in a form**

Save and run the project. Use the arrows on the data control to scroll through the data.

**Step5:** On any record, change the data in the author name or year born field. Move ahead, then move back to the record you changed. Note that your changes remain in effect. The data control automatically updates a record when you move off of the record.

**Note:-** This exercise demonstrated that you can create a simple but functional application that allows the user to browse through the rows of a database table (or result set) and to update rows in that table without writing any code.

---

## 10.14 RECORDSET

---

A recordset is a data structure that consists of a group of database records,



and can either come from a base table or as the result of a query to the table. The concept is common to a number of platforms, notably Microsoft's Data Access Objects (DAO) and ActiveX Data Objects (ADO). The Recordset object contains a Fields collection, and a Properties collection. At any time, the Recordset object refers to only a single record within the set as the current record. The method TMS uses for sending and receiving data to and from the database is a custom recordset object (cRecordset). Here, whenever recordset is mentioned i.e. referring to the custom cRecordset implementation, rather than to the various DAO/RDO/ADO incarnations.

A recordset is returned from the DAL by calling the OpenRecordset method. This recordset is completely disconnected from the data source. Updating a recordset will have no effect on the central data source until you call the UpdateRecordset method on the DAL. We could have easily returned an array of data from our DAL, especially since ADO and RDO nicely support GetRows, which does exactly that, but arrays are limited in a number of ways. Array manipulation is horrible. In Visual Basic, you can resize only the last dimension of an array, so forget about adding columns easily. Also, arrays are not self-documenting. Retrieving information from an array means relying on such hideous devices as constants for field names and the associated constant maintenance, or the low-performance method of looping through indexes looking for fields.

Enabling varying rows and columns involves using a data structure known as a ragged array-essentially an array of arrays-which can be cumbersome and counterintuitive to develop against. The advantage of using a custom recordset object is that we can present the data in a way that is familiar to most programmers, but we also get full control of what is happening inside the recordset. We can again simplify and customize its operation to support the rest of our components. Notice the Serialize method, which allows us to move these objects easily across machine boundaries. More on this powerful method later. For the moment, let's look at the typical interface of a cRecordset (Table 10.5).

**Table 10.5 cRecordset Interface**

<b>Member</b>	<b>Description</b>
MoveFirst	Moves to first record
MoveNext	Moves to next record
MovePrevious	Moves to previous record
MoveLast	Moves to last record
Name	Shows the name of the recordset
Fields	Returns a Field object

Synchronize	Refreshes the contents of the recordset
RowStatus	Shows whether this row has been created, updated, or deleted
RowCount	Shows the number of records in the recordset
AbsolutePosition	Shows the current position in the recordset
Edit	Copies the current row to a buffer for modification
AddNew	Creates an empty row in a buffer for modification
Update	Commits the modification in the buffer to the recordset
Serialize	Converts or sets the contents of the recordset to an array

---

## 10.15 APPLICATIONS OF DBMS

---

DBMS touches almost every aspect of real life though some important Database applications areas are given below:

- **Banking:** all transactions
- **Airlines:** reservations, schedules
- **Universities:** registration, grades
- **Sales:** customers, products, purchases
- **Manufacturing:** production, inventory, orders, supply chain
- **Human resources:** employee records, salaries, tax deductions

### CHECK YOUR PROGRESS

- How a table in SQL can be created?
- Compare hardware exception and software exception.
- Write any three applications of DBMS.

---

## 10.16 SUMMARY

---

A database is an organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex than they are often developed using formal design and modeling techniques. By data, we mean known facts that can be recorded and have embedded meaning too. Usually people use software such as Oracle, MS Access, or MS Excel to store data in the form of databases. A datum is a unit of data. Meaningful data combined to form information. Hence, information is basically interpreted data- data provided with semantics.

DBMS refers to the technology for creating and managing databases. It is a software tool to organize (create, retrieve, update and manage) data in a database. The main aim of a DBMS is to supply a way to store up and retrieve database information that is both convenient and efficient. DBMS system, stores data in either a navigational or hierarchical form. DBMS does not support the integrity constants. Examples of DBMS are a file system, XML, Windows Registry, etc.

The architecture of DBMS depends on the computer system on which it runs. For example, in a client-server DBMS architecture, the database systems at server machine can run several requests made by client machine. DBMS has the key components: Software, Data, Procedures, Database languages, Query processor, Runtime database manager, Database manager, Database engine, Reporting.

A DBMS can be used for- creation of a database, retrieval of information from the database, updating the database, managing a database. The main advantages of DBMS are – minimizing data redundancy, sharing of data, data consistency, data integrity, search capability, security, privacy, simplicity, backup and recovery, integrity constraints, data atomicity, development of new applications, concurrency control, data migration, tunability, solves enterprise and individual requirement, powerful user language, standards can be enforced, very less chances of data loss.

SQL (Structured Query Language), pronounced as "sequel", is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

A recordset is a data structure that consists of a group of database records, and can either come from a base table or as the result of a query to the table. The concept is common to a number of platforms, notably Microsoft's Data Access Objects (DAO) and ActiveX Data Objects (ADO). The Recordset object contains a Fields collection, and a Properties collection. At any time, the Recordset object refers to only a single record within the set as the current record.

Relational Database Management System (RDBMS) is an advanced version of the DBMS. The RDBMS uses a tabular structure where the headers are

the column names, and the rows contain corresponding values. RDBMS supports the integrity constraints at the schema level. Examples of RDBMS are- MySQL, Oracle, SQL Server, etc.

ActiveX Data Objects (ADO) is designed to be an easy-to-use application-level interface to any OLE DB data provider, including relational and non-relational databases, e-mail and file systems, text and graphics, and custom business objects, as well as existing ODBC data sources. Virtually all of the data available throughout the enterprise is available using the ADO data access technology.

A recordset is returned from the DAL by calling the OpenRecordset method. This recordset is completely disconnected from the data source. Updating a recordset will have no effect on the central data source until you call the UpdateRecordset method on the DAL.

---

## **10.17 TERMINAL QUESTIONS**

---

1. What do you understand by DBMS? Explain its need.
2. Explain the important components of DBMS.
3. Write a short note on advantages and disadvantages of DBMS.
4. Explain the responsibilities of DBA.
5. What do you understand by ODBC? Explain its architecture.
6. Describe the advantages and disadvantages of using DAO.
7. Explain the general characteristics of RDO.
8. What do you understand by SQL? Write the syntax for – creation of table, updation of table, inserting the records in a table.
9. Write a short note on recordset.
10. Discuss the main applications of DBMS.

---

# UNIT-11 NETWORK PROGRAMMING

---

## Structure

- 11.0 Introduction
- 11.1 Objectives
- 11.2 Introduction to Winsock
- 11.3 Windows Socket in General
- 11.4 Creating Sockets
- 11.5 Miscellaneous API
- 11.6 Winsock Catalog
- 11.7 Windows Objects
- 11.8 Access Control Story
- 11.9 Security Descriptors
- 11.10 Summary
- 11.11 Terminal Questions

---

## 11.0 INTRODUCTION

---

This unit primarily focuses on network programming and termed as Winsock programming. Winsock programming is basically Winsock application programming interface or Winsock API. If one wants to know about the windows network programming, then the first need to understand about Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Protocol i.e. IPV4 & IPV6. The IPV4 and IPV6 are basically intended with the multicasting.

Moreover, the unit also discusses about the Sockets. Sockets are the fundamental "things" behind any kind of network communications done by a computer. For example, when you type `www.google.com` in your web browser, it opens a socket and connects to `google.com` to fetch the page and shown it to you. Same thing happens with any chat clients like Gtalk or Skype. Generally, any network communication goes through a socket. The computer programming using socket is also called socket programming.

Winsock2.h is the header file to be included for winsock functions. `ws2_32.lib` is the library file to be linked with the program to be able to use winsock functions. The `WSAStartup` function is used to start or initialize winsock library. It takes two parameters; the first one is the version we want to load and second one is a `WSADATA` structure which will hold additional information after winsock has been loaded. If any error occurs, then the `WSAStartup` function

would return a non-zero value and `WSAGetLastError` can be used to get more information about what error happened.

---

## 11.1 OBJECTIVES

---

At the end of this unit you will come to know about:

- Concept of Winsock
- Windows Socket
- Creation of Sockets
- Winsock Catalog
- Windows Objects
- Access Control Story
- Security Descriptors

---

## 11.2 INTRODUCTION TO WINSOCK

---

Winsock is a standard application programming interface (API) that allows two or more applications (or processes) to communicate either on the same machine or across a network and is primarily designed to foster data communication over a network. It is important to understand that Winsock is a network programming interface and not a protocol. Winsock provides the programming interface for applications to communicate using popular network protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP) and Internetwork Packet Exchange (IPX). The Winsock interface inherits a great deal from the BSD Sockets implementation on UNIX platforms. In Windows environments, the interface has evolved into a truly protocol-independent interface, especially with the release of Winsock 2.

You can differentiate the two functions with the WSA prefix. If Winsock 2 updated or added a new API function in its specification, the function name is prefixed with WSA. For example, the Winsock 1 function to create a socket is simply `socket()`. Winsock 2 introduces a newer version named `WSASocket()` that is capable of using some of the new features made available in Winsock 2. There are a few exceptions to this naming rule. `WSAStartup()`, `WSACleanup()`, `WSARecvEx()`, and `WSAGetLastError()` are in the Winsock 1.1 specification.

---

### 11.2.1 WINSOCK SYSTEM ARCHITECTURE

---

The majority of the Winsock API is implemented in `WS2_32.DLL` and is declared in `WINSOCK2.H`. The only exception is for the Microsoft-specific Winsock extensions (such as `TransmitFile`, `AcceptEx`, etc.), which are located in `MSWSOCK.DLL`. These extensions are not a part of the formal Winsock specification but have been added by Microsoft. Also, because these are Microsoft-specific extensions and are not part of the formal Winsock specification, some of the extension APIs are available only on certain versions of

Windows.

When an application calls into the Winsock API, it calls into WS2\_32.DLL. The Winsock DLL performs some parameter validation and then determines which protocol service provider the call should be routed to. There may be multiple providers installed in the Winsock catalog and WS2\_32.DLL determines which provider should handle the call.

There are two types of providers — base and layered. A base provider sits on top of a transport protocol, such as Microsoft TCP/IP and UDP/IP providers or the Resource Reservation Protocol (RSVP) provider, which implements QOS. The Microsoft base provider consists of MSAFD.DLL and MSWSOCK.DLL, but actually exposes one or more providers for the individual protocols of TCP/IP, IPX/SPX, NetBIOS, AppleTalk, etc. A layered provider sits below WS2\_32.DLL and above a base provider and can intercept and manipulate the Winsock calls. That is, if an application creates a socket from the layered provider, the layered provider will intercept all Winsock calls using that socket. The layered provider may block, modify, or pass the call unmodified to the underlying provider. Also, there may be numerous layered providers installed, one on top of another.

Once a Winsock call makes it to the base provider, the base provider will in turn make calls to the Winsock kernel mode component. Unlike some other operating systems, the Windows NT transport protocols do not have a Winsock-like interface that applications can use to directly talk to them. Instead, they implement a much more general API called the Transport Driver Interface (TDI). The generality of this API allows the Windows NT subsystems to free themselves from being tied to a particular version-of-the-decade network-programming interface. The sockets emulation is provided by the Winsock kernel mode driver (currently implemented in AFD.SYS). This driver is responsible for the connection and buffer management related to providing a sockets-like interface to an application. AFD, in turn, talks TDI to the transport protocol driver as shown in Fig 11.1.

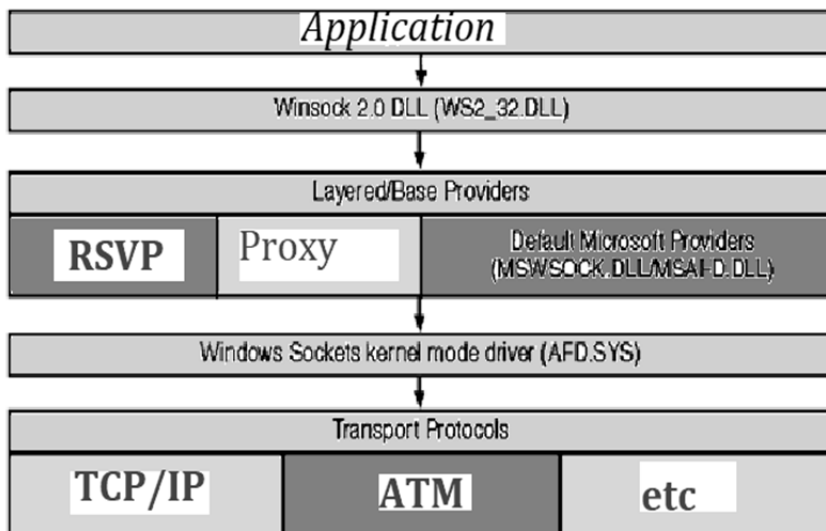


Fig. 11.1 Winsock System Architecture

---

## 11.2.2 WINSOCK HEADERS AND LIBRARIES

---

When developing new application, you should target the Winsock 2 specification by including WINSOCK2.H in your application. For compatibility with older Winsock applications and when developing on Windows CE platforms, WINSOCK.H is available. There is also an additional header file—MSWSOCK.H, which targets Microsoft-specific programming extensions that are normally used for developing high performance Winsock applications.

When compiling your application with WINSOCK2.H, you should link with WS2\_32.LIB library. When using WINSOCK.H (as on Windows CE) you should use WSOCK32.LIB. If you use extension APIs from MSWSOCK.H, you must also link with MSWSOCK.LIB. Once you have included the necessary header files and link environment, you are ready to begin coding your application, which requires initializing Winsock.

---

## 11.2.3 INITIALIZING WINSOCK

---

Every Winsock application must load the appropriate version of the Winsock DLL. If you fail to load the Winsock library before calling a Winsock function, the function returns a SOCKET\_ERROR; the error will be WSANOTINITIALISED. Loading the Winsock library is accomplished by calling the WSASStartup() function, which is defined as:

```
int WSASStartup(  
    WORD wVersionRequested,  
    LPWSADATA lpWSADATA  
);
```

The wVersionRequested parameter is used to specify the version of the Winsock library you want to load. The high-order byte specifies the minor version of the requested Winsock library, while the low-order byte is the major version. You can use the handy macro MAKEWORD(x, y), in which x is the high byte and y is the low byte, to obtain the correct value for wVersionRequested. The lpWSADATA parameter is a pointer to a LPWSADATA structure that WSASStartup() fills with information related to the version of the library it loads:

```
typedef struct WSADATA  
{  
    WORD          wVersion;  
    WORD          wHighVersion;  
    char          szDescription[WSADESCRIPTION_LEN + 1];  
    char          szSystemStatus[WSASYS_STATUS_LEN + 1];  
    unsigned short iMaxSockets;  
    unsigned short iMaxUdpDg;
```



```
char FAR *      lpVendorInfo;
} WSADATA, * LPWSADATA;
```

WSAStartup() sets the first field, wVersion, to the Winsock version you will be using. The wHighVersion parameter holds the highest version of the Winsock library available. Remember that in both of these fields, the high-order byte represents the Winsock minor version, and the low-order byte is the major version. The szDescription and szSystemStatus fields are set by the particular implementation of Winsock and aren't really useful. Do not use the next two fields, iMaxSockets and iMaxUdpDg. They are supposed to be the maximum number of concurrently open sockets and the maximum datagram size; however, to find the maximum datagram size you should query the protocol information through WSAEnumProtocols(). The maximum number of concurrent sockets isn't some magic number, it depends more on the physical resources available. Finally, the lpVendorInfo field is reserved for vendor-specific information regarding the implementation of Winsock. This field is not used on any Windows platforms.

For the most part, when writing new applications, you would load the latest version of the Winsock library currently available. Remember that if, for example, Winsock 3 is released, your application that loads version 2.2 should run as expected. If you request a Winsock version later than that which the platform supports, WSAStartup() will fail. Upon return, the wHighVersion of the WSADATA structure will be the latest version supported by the library on the current system. When your application is completely finished using the Winsock interface, you should call WSACleanup(), which allows Winsock to free up any resources allocated by Winsock and cancel any pending Winsock calls that your application made. WSACleanup() is defined as:

```
int WSACleanup(void);
```

Failure to call WSACleanup when your application exits is not harmful because the operating system will free up resources automatically; however, your application will not be following the Winsock specification. Also, you should call WSACleanup for each call that is made to WSAStartup.

---

## 11.2.4 ERROR CHECKING AND HANDLING

---

We'll first cover error checking and handling, as they are vital to writing a successful Winsock application. It is actually common for Winsock functions to return an error; however, there are some cases in which the error is not critical and communication can still take place on that socket. The most common return value for an unsuccessful Winsock call is SOCKET\_ERROR, although this is certainly not always the case. When covering each API call in detail, we'll point out the return value corresponding to an error. The constant SOCKET\_ERROR actually is -1. If you make a call to a Winsock function and an error condition occurs, you can use the function WSAGetLastError() to obtain a code that indicates specifically what happened. This function is defined as:

```
int WSAGetLastError (void);
```

A call to the function after an error occurs will return an integer code for

the particular error that occurred. These error codes returned from `WSAGetLastError()` all have predefined constant values that are declared in either `WINSOCK.H` or `WINSOCK2.H`, depending on the version of Winsock. The only difference between the two header files is that `WINSOCK2.H` contains more error codes for some of the newer API functions and capabilities introduced in Winsock 2. The constants defined for the various error codes (with `#define` directives) generally begin with `WSAE`. On the flip side of `WSAGetLastError()`, there is `WSASetLastError()`, which allows you to manually set error codes that `WSAGetLastError()` retrieves.

The following program demonstrates how to construct a skeleton Winsock application based on the discussion so far:

```
#include <winsock2.h>

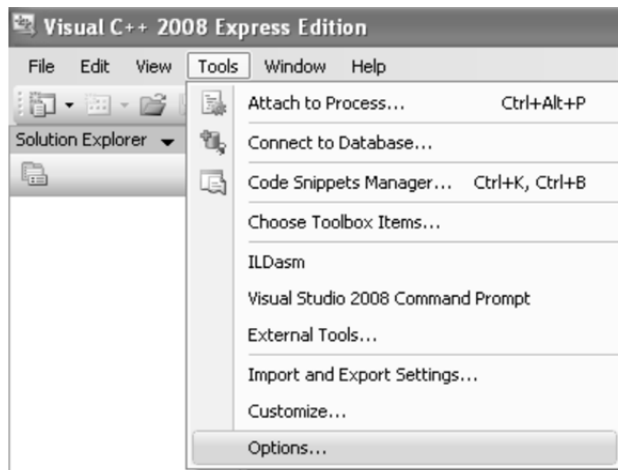
void main(void)
{
    WSADATA wsaData;

    // Initialize Winsock version 2.2
    if ((Ret = WSASStartup(MAKEWORD(2,2), &wsaData)) != 0)
    {
        printf("WSASStartup failed with error %ld\n",
            WSAGetLastError());
        return;
    }

    // Setup Winsock communication code here

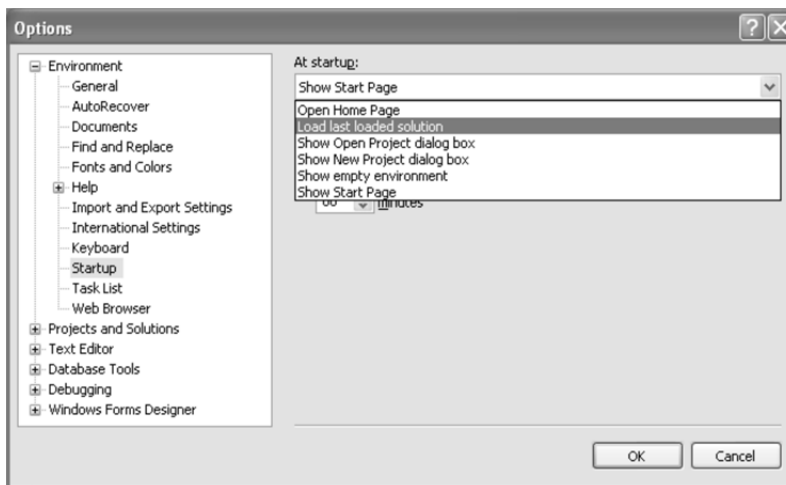
    // When your application is finished call WSACleanup
    if (WSACleanup() == SOCKET_ERROR)
    {
        printf("WSACleanup failed with error %d\n",
            WSAGetLastError());
    }
}
```

Let's try this program using Visual C++ 2008 Express Edition. First and foremost, let change the newly installed VC++ startup page to last loaded solution. You can skip this 'optional' step. Click Tools menu > Options sub menu (Fig.11.2).



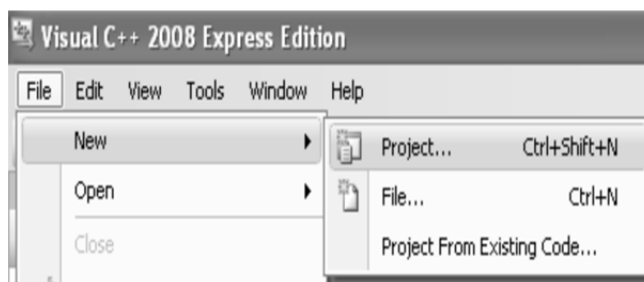
**Fig. 11.2 Tools menu**

Expand Environment folder > Select Startup link > Set the At Startup: to Load last loaded solution > Click OK (Fig. 11).



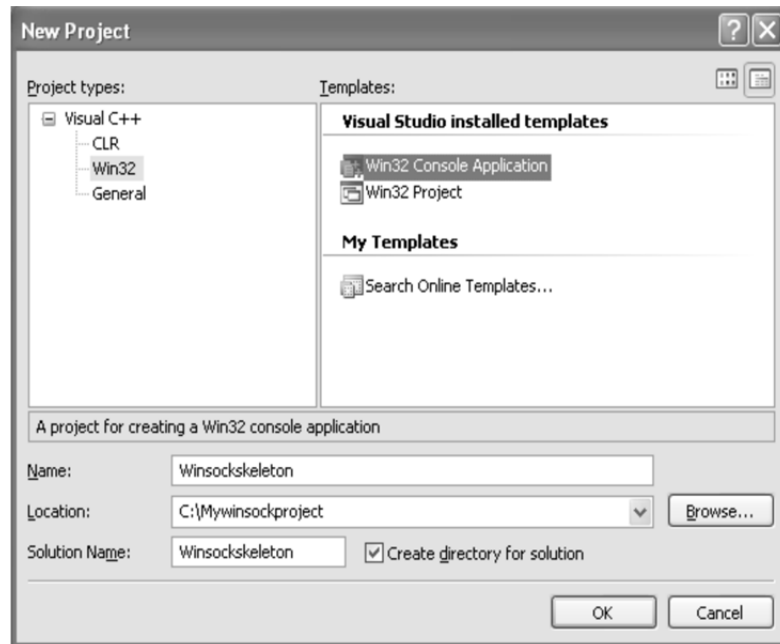
**Fig. 11.3 Options- load last loaded options**

1. Then we can start creating the Win32 console application project. Click File menu > Project sub menu to create a new project (Fig. 11.4).



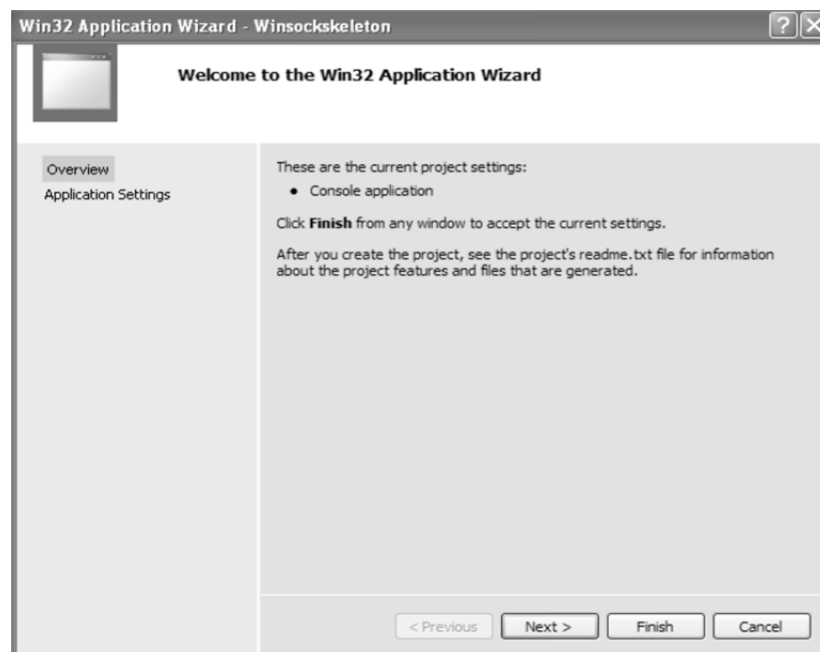
**Fig. 11.4 Creating a new project**

2. Select Win32 for the Project types, and Win32 Console Application for the Templates, Put the project and solution name. Adjust the project location if needed and click OK (Fig. 11.5).



**Fig. 11.5 Win32 Console Application**

3. Click Next for the Win32 Application Wizard Overview page. We will remove all the unnecessary project items (Fig. 11.6).



**Fig. 11.6 Application Wizard**

- In the Application page, select Empty project for the Additional options: Leave others as given and click Finish (Fig. 11.7).

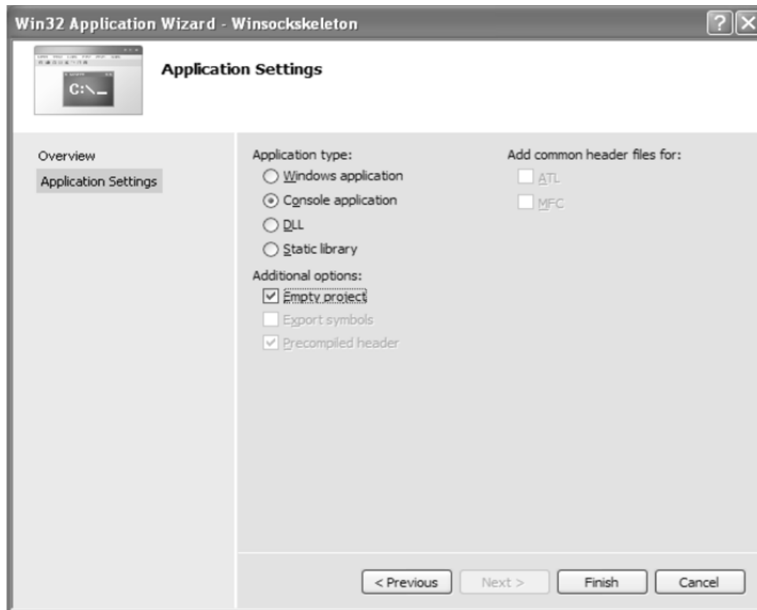


Fig. 11.7 Application Settings

- Next, we need to add new source file. Click Project menu > Add New Item sub menu or select the project folder in the Solution Explorer > Select Add menu > Select New Item sub menu (Fig. 11.8).

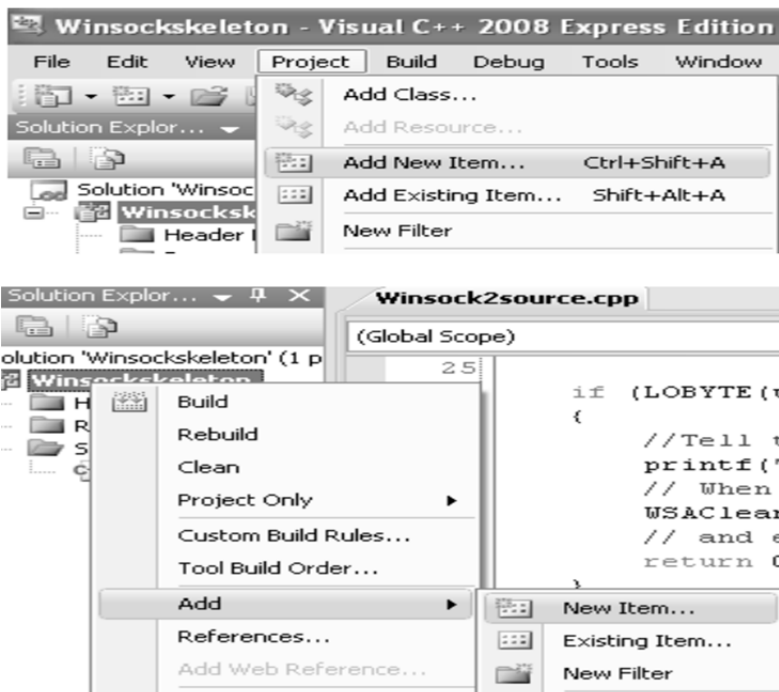
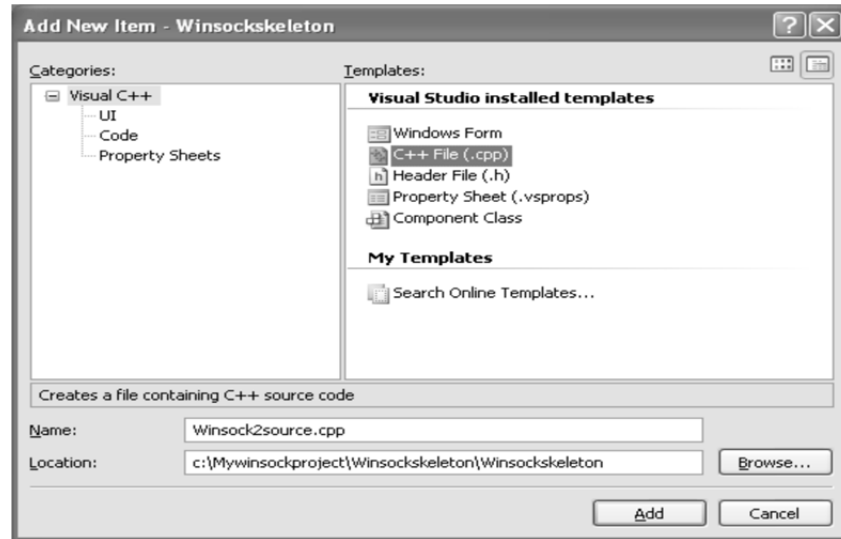


Fig. 11.8 Wssockskeleton

6. Select C++ File (.cpp) for the Templates: Put the source file name and click Add. Although the extension is .cpp, Visual C++ IDE will recognize that the source code used is C based on the Compile as C Code (/TC) option which will be set in the project property page later (Fig. 11.9).



**Fig. 11.9 Adding new item**

7. Now, add the source code as given below.

```
#include <winsock2.h>
#include <stdio.h>

int main(void)
{
    WSADATA wsaData;
    int RetCode;

    // Initialize Winsock version 2.2
    if ((RetCode= WSStartup(MAKEWORD(2,2), &wsaData)) != 0)
    {
        printf("WSStartup failed with error %d\n", RetCode);
        return 1;
    }
    else
    {
```

```

        printf("The Winsock dll found!\n");
        printf("The current status is: %s.\n",
            wsaData.szSystemStatus);
    }
if(LOBYTE(wsaData.wVersion)!=2||HIBYTE(wsaData.wVersion)!= 2 )
    {
//Tell the user that we could not find a usable WinSock DLL
printf("The dll do not support the Winsock version
%u.%u!\n",
        LOBYTE(wsaData.wVersion),HIBYTE(wsaData.wVersion));
// When your application is finished call WSACleanup

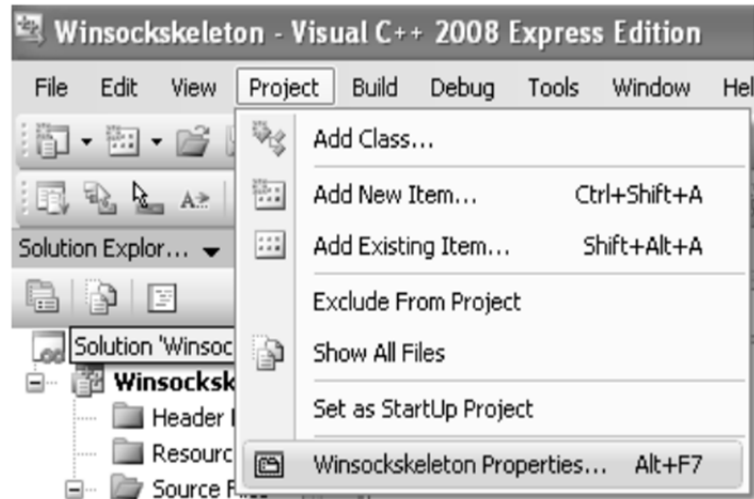
WSACleanup();
        // and exit
        return 0;
    }
    else
    {
printf("The dll supports the Winsock version %u.%u!\n",

LOBYTE(wsaData.wVersion),HIBYTE(wsaData.wVersion));
printf("The highest version this dll can support: %u.%u\n",
LOBYTE(wsaData.wHighVersion),
HIBYTE(wsaData.wHighVersion));

        // Setup Winsock communication code here
// When your application is finished call WSACleanup
        if (WSACleanup() == SOCKET_ERROR)
printf("WSACleanup failed with error %d\n",
WSAGetLastError());
        // and exit
        return 1;
    }
}

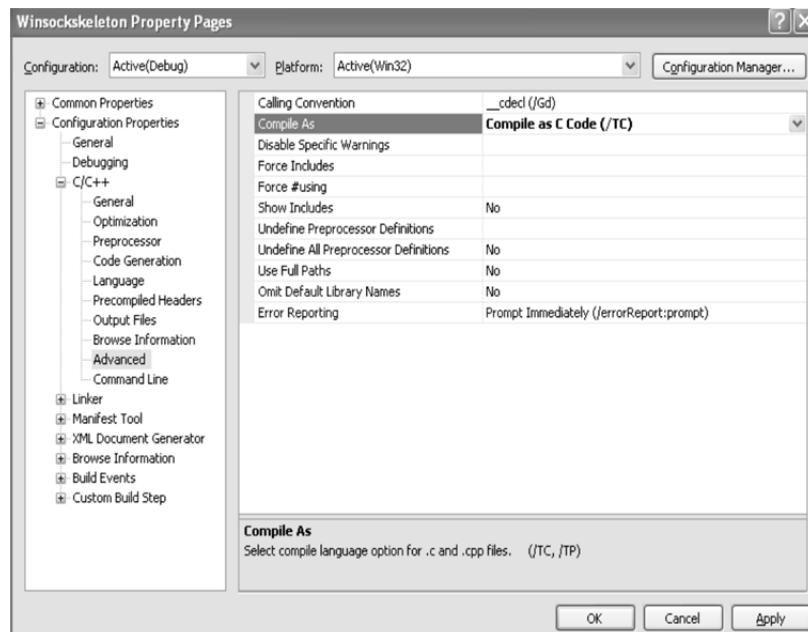
```

we need to set the project to be compiled as C code and link to ws2\_32.lib, the Winsock2 library. Invoke the project property page (Fig. 11.10).



**Fig. 11.10 Invoking project property**

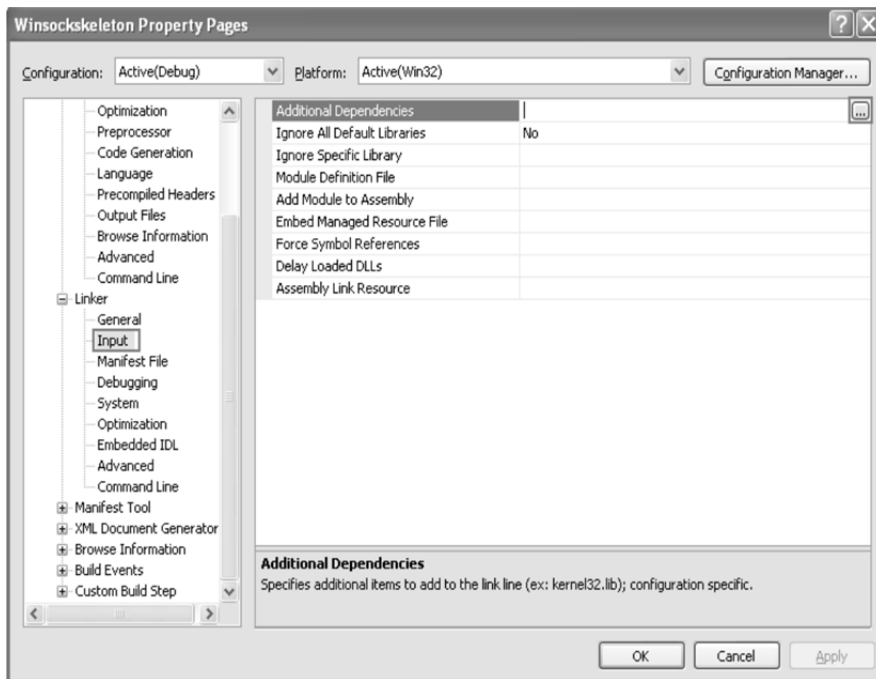
9. Expand the Configuration folder > Expand the C/C++ sub folder. Select the Advanced link and for the Compile As option, select Compile as C Code (/TC) (Fig. 11.11).



**Fig. 11.11 Winsocskelton property pages**

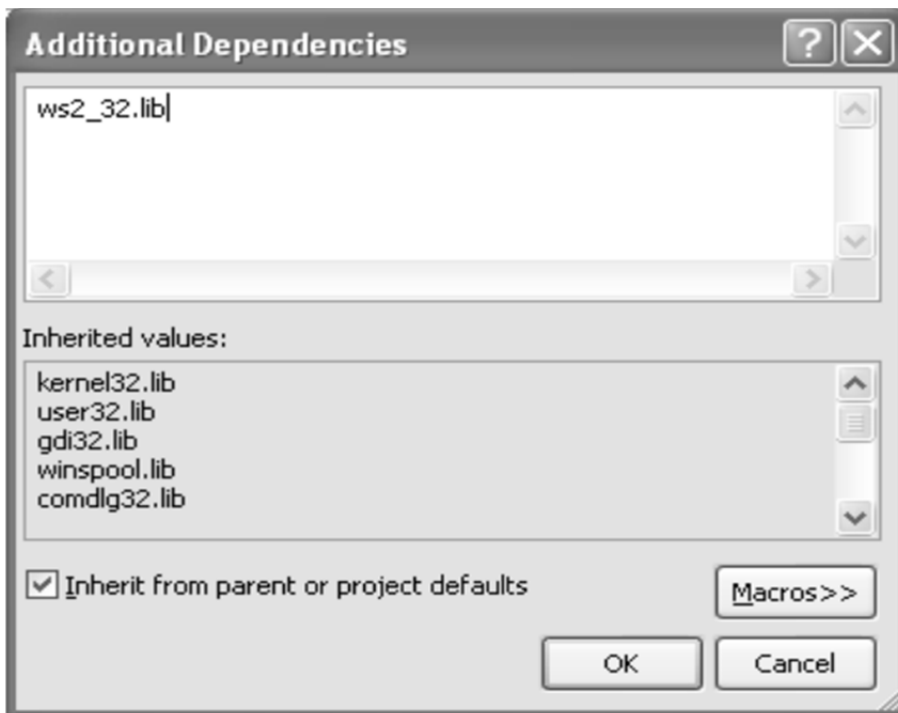
10. Next, expand the Linker folder and select the Input link. For the Additional Dependencies option, click the ellipses at the end of the empty field on the right side (Fig. 11.12).





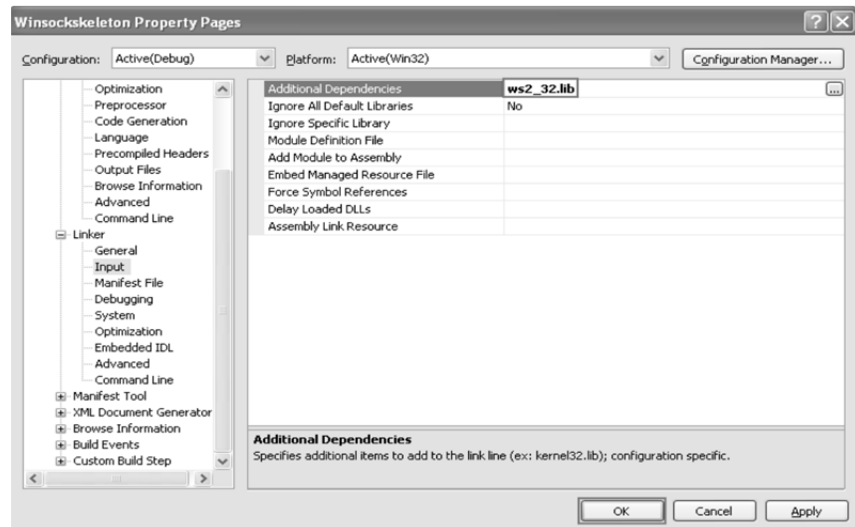
**Fig. 11.12 Selecting the input link**

11. Manually, type the library name and click OK (Fig. 11.13).



**Fig. 11.13 Typing the library**

12. Or you can just directly type the library name in the empty field on the right of the Additional Dependencies, Click OK (Fig. 11.14).



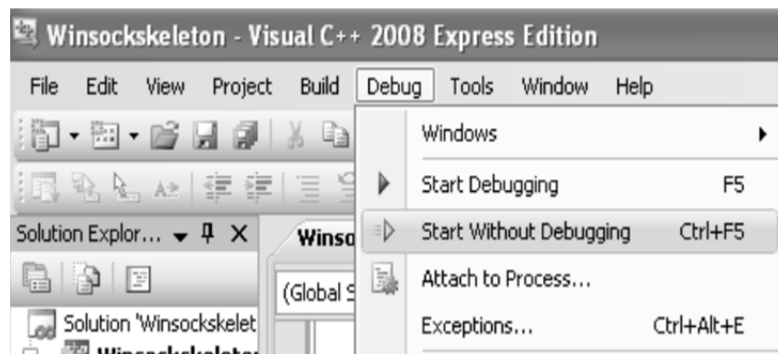
**Fig. 11.14 Additional dependencies**

13. Build the project and make sure there is no error which can be seen (if any) in the Output window normally docked at the bottom of the IDE by default (Fig. 11.15).



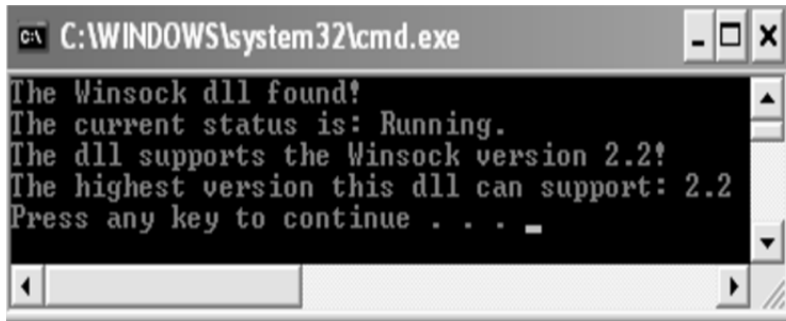
**Fig. 11.15 Building the solution**

14. Run the project (Fig. 11.16).



**Fig. 11.16 Starting without debugging**

15. If there is no error, a sample of expected output is shown below (Fig. 11.7).



```
C:\WINDOWS\system32\cmd.exe
The Winsock dll found!
The current status is: Running.
The dll supports the Winsock version 2.2!
The highest version this dll can support: 2.2
Press any key to continue . . . -
```

**Fig. 11.17 Output**

Well, after completing this exercise you should be familiar with the steps to create an empty Win32 console application project. Those steps will be repeated for almost all the Winsock2 projects in this tutorial.

---

## 11.3 WINDOWS SOCKET IN GENERAL

---

The Windows Sockets specification defines a binary-compatible network programming interface for Microsoft Windows. Windows Sockets are based on the UNIX sockets implementation in the Berkeley Software Distribution (BSD) from the University of California at Berkeley. The specification includes both BSD-style socket routines and extensions specific to Windows. Using Windows Sockets permits your application to communicate across any network that conforms to the Windows Sockets API. On Win32, Windows Sockets provide for thread safety.

Many network software vendors support Windows Sockets under network protocols including TCP/IP, Xerox Network System (XNS), Digital Equipment Corporation's DECNet protocol, Novell Corporation's Internet Packet Exchange/Sequenced Packed Exchange (IPX/SPX), and others. Although the present Windows Sockets specification defines the sockets abstraction for TCP/IP, any network protocol can comply with Windows Sockets by supplying its own version of the dynamic link library (DLL) that implements Windows Sockets. Examples of commercial applications written with Windows Sockets include X Window servers, terminal emulators, and electronic mail systems.

**Note:-** The purpose of Windows Sockets is to abstract away the underlying network so you don't have to be knowledgeable about that network and so your application can run on any network that supports sockets.

The Microsoft Foundation Class Library (MFC) supports programming with the Windows Sockets API by supplying two classes. One of these classes, CSocket, provides a high level of abstraction to simplify your network communications programming.

The Windows Sockets specification, Windows Sockets: An Open

Interface for Network Computing Under Microsoft Windows, now at version 1.1, was developed as an open networking standard by a large group of individuals and corporations in the TCP/IP community and is freely available for use. The sockets programming model supports one “communication domain” currently, using the Internet Protocol Suite. The specification is available in the Win32 SDK.

**Note:-** Because sockets use the Internet Protocol Suite, they are the preferred route for applications that support Internet communications on the “information highway.”

---

### 11.3.1 DEFINITION OF A SOCKET

---

A socket is a communication endpoint — an object through which a Windows Sockets application sends or receives packets of data across a network. A socket has a type and is associated with a running process, and it may have a name. Currently, sockets generally exchange data only with other sockets in the same “communication domain,” which uses the Internet Protocol Suite.

Both kinds of sockets are bi-directional— they are data flows that can be communicated in both directions simultaneously (full-duplex).

Two socket types are available:

- **Stream sockets**— Stream sockets provide for a data flow without record boundaries i.e. a stream of bytes. Streams are guaranteed to be delivered and to be correctly sequenced (means that packets are delivered in the order sent) and unduplicated (means that you get a particular packet only once).
- **Datagram sockets**— Datagram sockets support a record-oriented data flow that is not guaranteed to be delivered and may not be sequenced as sent or unduplicated.

**Note:-** Under some network protocols, such as XNS, streams can be record-oriented— streams of records rather than streams of bytes. Under the more common TCP/IP protocol, however, streams are byte streams. Windows Sockets provides a level of abstraction independent of the underlying protocol.

---

### 11.3.2 THE SOCKET DATA TYPE

---

Each MFC socket object encapsulates a handle to a Windows Sockets object. The data type of this handle is SOCKET. A SOCKET handle is analogous to the HWND for a window. MFC socket classes provide operations on the encapsulated handle.

---

### 11.3.3 USES FOR SOCKETS

---

Sockets are highly useful in at least three communications contexts:

- Client/Server models

- Peer-to-peer scenarios, such as chat applications
- Making remote procedure calls (RPC) by having the receiving application interpret a message as a function call

### **CHECK YOUR PROGRESS**

- Define the term Winsock.
- What do you understand by error checking and handling?
- Describe stream sockets.

---

## **11.4 CREATING A SOCKET**

---

If you're familiar with Winsock, you know that the API is based on the concept of a socket. A socket is a handle to a transport provider. In Windows, a socket is not the same thing as a file descriptor and therefore is a separate type: `SOCKET` in `WSOCK2.H`. There are two functions that can be used to create a socket: `socket` and `WSASocket`.

### **Syntax:**

```
SOCKET socket(int af, int type, int protocol);
```

The first parameter, `af`, is the protocol's address family. Since we describe Winsock in this Unit using only the IPv4 protocol, you should set this field to `AF_INET`. The second parameter, `type`, is the protocol's socket type. When you are creating a socket to use TCP/IP, set this field to `SOCK_STREAM`, for UDP/IP use `SOCK_DGRAM`. The third parameter is `protocol` and is used to qualify a specific transport if there are multiple entries for the given address family and socket type. For TCP you should set this field to `IPPROTO_TCP`; for UDP use `IPPROTO_UDP`.

There are four useful Winsock functions to control various socket options and socket behaviors — `setsockopt`, `getsockopt`, `ioctlsocket`, and `WSAIoctl`. For simple Winsock programming, you will not need to use them specifically. Once you have successfully created a socket, you are ready to set up communication on the socket to prepare it for sending and receiving data. In Winsock there are two basic communication techniques: connection-oriented and connectionless communication.

---

### **11.4.1 CONNECTION-ORIENTED COMMUNICATION**

---

A connection-oriented service is one that establishes a dedicated connection between the communicating entities before data communication commences. It is modeled after the telephone system. To use a connection-oriented service, the user first establishes a connection, uses it and then releases it. In connection-oriented services, the data streams/packets are delivered to the receiver in the same order in which they have been sent by the sender.

Connection-oriented services may be done in either of the following ways –

- **Circuit-switched connection:** In circuit switching, a dedicated physical path or a circuit is established between the communicating nodes and then data stream is transferred.
- **Virtual circuit-switched connection:** Here, the data stream is transferred over a packet switched network, in such a way that it seems to the user that there is a dedicated path from the sender to the receiver. A virtual path is established here. However, other connections may also be using this path.

Connection-oriented services may be of the following types –

- Reliable Message Stream: e.g. sequence of pages
- Reliable Byte Stream: e.g. song download
- Unreliable Connection: e.g. VoIP (Voice over Internet Protocol)

In IP, connection-oriented communication is accomplished through the TCP/IP protocol. TCP provides reliable error-free data transmission between two computers. When applications communicate using TCP, a virtual connection is established between the source computer and the destination computer. Once a connection is established, data can be exchanged between the computers as a two-way stream of bytes.

---

## 11.4.2 CONNECTIONLESS COMMUNICATION

---

A Connectionless service is a data communication between two nodes where the sender sends data without ensuring whether the receiver is available to receive the data. Here, each data packet has the destination address and is routed independently irrespective of the other packets. Thus the data packets may follow different paths to reach the destination. There's no need to setup connection before sending a message and relinquish it after the message has been sent. The data packets in a connectionless service are usually called datagrams.

Protocols for connectionless services are –

- Internet Protocol (IP)
- User Datagram Protocol (UDP)
- Internet Control Message Protocol (ICMP)

Connectionless services may be of the following types –

- A datagram with Acknowledgement: e.g. text messages with delivery report
- Request-Reply: e.g. queries from remote databases

Connectionless communication behaves differently than connection-oriented communication, so the method for sending and receiving data is substantially different. In IP, connectionless communication is accomplished

through UDP/IP. UDP doesn't guarantee reliable data transmission and is capable of sending data to multiple destinations and receiving it from multiple sources. For example, if a client sends data to a server, the data is transmitted immediately regardless of whether the server is ready to receive it. If the server receives data from the client, it doesn't acknowledge the receipt. Data is transmitted using datagrams, which are discrete message packets.

---

## 11.5 MISCELLANEOUS API

---

In this section, few Winsock API functions have been covered that you might find useful when you put together your own network applications.

---

### 11.5.1 GETPEERNAME

---

This function is used to obtain the peer's socket address information on a connected socket. The function is defined as:

```
int getpeername(  
    SOCKET s,  
    struct sockaddr FAR* name,  
    int FAR* namelen  
    );
```

The first parameter is the socket for the connection; the last two parameters are a pointer to a SOCKADDR structure of the underlying protocol type and its length. For datagram sockets, this function returns the address passed to a connect call; however, it will not return the address passed to a sendto or WSASendTo call.

---

### 11.5.2 GETSOCKNAME

---

This function is the opposite of getpeername. It returns the address information for the local interface of a given socket. The function is defined as follows:

```
int getsockname(  
    SOCKET s,  
    struct sockaddr FAR* name,  
    int FAR* namelen  
    );
```

The parameters are the same as the getpeername parameters except that the address information returned for socket s is the local address information. In the case of TCP, the address is the same as the server socket listening on a specific port and IP interface.

---

### 11.5.3 WSADUPLICATESOCKET

---

The `WSADuplicateSocket` function is used to create a `WSAPROTOCOL_INFO` structure that can be passed to another process, thus enabling the other process to open a handle to the same underlying socket so that it too can perform operations on that resource. Note that this is necessary only between processes; threads in the same process can freely pass the socket descriptors. This function is defined as:

```
int WSADuplicateSocket(  
    SOCKET s,  
    DWORD dwProcessId,  
    LPWSAPROTOCOL_INFO lpProtocolInfo  
);
```

The first parameter is the socket handle to duplicate. The second parameter, `dwProcessId`, is the process ID of the process that intends to use the duplicated socket. Third, the `lpProtocolInfo` parameter is a pointer to a `WSAPROTOCOL_INFO` structure that will contain the necessary information for the target process to open a duplicate handle. Some form of interprocess communication must occur so that the current process can pass the `WSAPROTOCOL_INFO` structure to the target process, which then uses this structure to create a handle to the socket (using the `WSASocket` function).

Both socket descriptors can be used independently for I/O. Winsock provides no access control, however, so it is up to the programmer to enforce some kind of synchronization. All of the state information associated with a socket is held in common across all the descriptors because the socket descriptors are duplicated, not the actual socket. For example, any socket option set by the `setsockopt` function on one of the descriptors is subsequently visible using the `getsockopt` function from any or all descriptors. If a process calls `closesocket` on a duplicated socket, it causes the descriptor in that process to become deallocated. The underlying socket, however, will remain open until `closesocket` is called on the last remaining descriptor.

---

## 11.6 WINSOCK CATALOG

---

The Winsock catalog is a database that contains the different protocols available on the system. Winsock provides a method for determining which protocols are installed on a given workstation and returning a variety of characteristics for each protocol. If a protocol is capable of multiple behaviors, each distinct behavior type has its own catalog entry within the system. For example, if you install TCP/IP on your system, there will be two IP entries: one for TCP, which is reliable and connection-oriented, and one for UDP, which is unreliable and connectionless.

The function call to obtain information on installed network protocols is `WSAEnumProtocols` and is defined as:



```

int WSAEnumProtocols (
    LPINT lpiProtocols,
    LPWSAPROTOCOL_INFO lpProtocolBuffer,
    LPDWORD lpdwBufferLength
);

```

This function supersedes the Winsock 1.1 function EnumProtocols, the necessary function for Windows CE. The only difference is that WSAEnumProtocols returns an array of WSAPROTOCOL\_INFO structures, whereas EnumProtocols returns an array of PROTOCOL\_INFO structures that contain fewer fields than the WSAPROTOCOL\_INFO structure (but more or less the same information).

The WSAPROTOCOL\_INFO structure is defined as

```

typedef struct _WSAPROTOCOL_INFO {
    DWORD          dwServiceFlags1;
    DWORD          dwServiceFlags2;
    DWORD          dwServiceFlags3;
    DWORD          dwServiceFlags4;
    DWORD          dwProviderFlags;
    GUID           ProviderId;
    DWORD          dwCatalogEntryId;
    WSAPROTOCOLCHAIN ProtocolChain;
    int            iVersion;
    int            iAddressFamily;
    int            iMaxSockAddr;
    int            iMinSockAddr;
    int            iSocketType;
    int            iProtocol;
    int            iProtocolMaxOffset;
    int            iNetworkByteOrder;
    int            iSecurityScheme;
    DWORD          dwMessageSize;
    DWORD          dwProviderReserved;
    TCHAR          szProtocol[WSAPROTOCOL_LEN + 1];
} WSAPROTOCOL_INFO, FAR * LPWSAPROTOCOL_INFO;

```

The easiest way to call WSAEnumProtocols is to make the first call with

BCA-118/313

lpProtocolBuffer equal to NULL and set lpdwBufferLength to 0. The call fails with WSAENOBUFS, but lpdwBufferLength then contains the correct size of the buffer required to return all the protocol information. Once you allocate the correct buffer size and make another call with the supplied buffer, the function returns the number of WSAPROTOCOL\_INFO structures returned. At this point, you can step through the structures to find the protocol entry with your required attributes. The sample program called ENUMCAT.C on the companion CD enumerates all installed protocols and prints out each protocol's characteristics.

**Note:-** On 64-bit Windows, it is possible to run 32-bit applications under the WOW64 (Windows on Windows) subsystem. Because both 32-bit and 64-bit applications may need to access the Winsock catalog, the system maintains two separate catalogs. When a 64-bit Winsock application runs and calls WSAEnumProtocols, the 64-bit catalog is used. Likewise, when a 32-bit Winsock application calls WSAEnumProtocols, the 32-bit catalog is used. This will become more important when dealing with the Winsock Service Provider Interface.

### **CHECK YOUR PROGRESS**

- Compare connection oriented and connection less communication.
- What for getpeername function is used?
- What do you understand Winsock catalog?

---

## **11.7 WINDOWS OBJECTS**

---

MFC supplies class CWnd to encapsulate the HWND handle of a window. The CWnd object is a C++ window object, distinct from the HWND that represents a Windows window but containing it. Use CWnd to derive your own child window classes, or use one of the many MFC classes derived from CWnd. Class CWnd is the base class for all windows, including frame windows, dialog boxes, child windows, controls, and control bars such as toolbars. A good understanding of the relationship between a C++ window object and an HWND is crucial for effective programming with MFC.

MFC provides some default functionality and management of windows, but you can derive your own class from CWnd and use its member functions to customize the provided functionality. You can create child windows by constructing a CWnd object and calling its Create member function, then customize the child windows using CWnd member functions. You can embed objects derived from CView, such as form views or tree views, in a frame window. And you can support multiple views of your documents via splitter panes, supplied by class CSplitterWnd. Each object derived from class CWnd contains a message map, through which you can map Windows messages or command IDs to your own handlers. The general literature on programming for Windows is a good resource for learning how to use the CWnd member functions, which encapsulate the HWND APIs.

---

## 11.7.1 FUNCTIONS FOR OPERATING ON A CWND

---

CWnd and its derived window classes provide constructors, destructors, and member functions to initialize the object, create the underlying Windows structures, and access the encapsulated HWND. CWnd also provides member functions that encapsulate Windows APIs for sending messages, accessing the window's state, converting coordinates, updating, scrolling, accessing the Clipboard, and many other tasks. Most Windows window-management APIs that take an HWND argument are encapsulated as member functions of CWnd. The names of the functions and their parameters are preserved in the CWnd member function. For details about the Windows APIs encapsulated by CWnd, see class CWnd.

---

## 11.7.2 CWND AND WINDOWS MESSAGES

---

One of the primary purposes of CWnd is to provide an interface for handling Windows messages, such as WM\_PAINT or WM\_MOUSEMOVE. Many of the member functions of CWnd are handlers for standard messages — those beginning with the identifier `afx_msg` and the prefix "On," such as `OnPaint` and `OnMouseMove`. Message Handling and Mapping covers messages and message handling in detail. The information there applies equally to the framework's windows and those that you create yourself for special purposes.

---

## 11.8 ACCESS CONTROL STORY

---

In this Module and that follows, we try to learn how the security is implemented in Windows Operating Systems. Access Control is one of the important and fundamental topics in Windows SDK Platform from Security point of view. We would start with the access control model used by Windows OSes and then dig deeper the details of every component in the model. On the way we will also be introduced with functions that available for manipulation and interaction with various objects of the Windows OS in the security aspect.

---

### 11.8.1 ACCESS CONTROL

---

It has been mentioned in MSDN documentation that at the beginning, Windows OSes follow already an obsolete Class C2 standard, formally known as Trusted Computer System Evaluation Criteria (TCSEC) then superseded by Common Criteria and the ISO version is ISO 15408 Common Criteria for Information Technology Security Evaluation (Part 1, 2 and 3). Access control refers to security features that control who can access which resources in the operating system. Applications call access control functions to set who can access specific resources or control access to resources provided by the applications.

---

### 11.8.2 ACCESS CONTROL MODEL

---

The access control model enables you to control the ability of a process to

access securable objects or to perform various system administration tasks. A process is a security context under which an application runs. Typically, the security context is associated with a user, so all applications running under a given process take on the permissions and privileges of the owning user.

---

### **11.8.3 ACCESS CONTROL COMPONENTS**

---

There are two basic components of the access control model:

1. Access tokens, which contain information about a logged-on user.
2. Security descriptors, which contain the security information that protects a securable object.

When a user logs on, the system authenticates the user's account name and password. If the logon is successful, the system creates an access token. Every process executed on behalf of this user will have a copy of this access token. The access token contains security identifiers (SID) that identify the user's account and any group accounts to which the user belongs. The token also contains a list of the privileges held by the user or the user's groups. The system uses this token to identify the associated user when a process tries to access a securable object or perform a system administration task that requires privileges.

When a securable object is created, the system assigns it a security descriptor that contains security information specified by its creator, or default security information if none is specified. Applications can use functions to retrieve and set the security information for an existing object. A security descriptor identifies the object's owner and can also contain the following access control lists (ACLs):

1. A discretionary access control list (DACL) that identifies the users and groups allowed or denied access to the object.
2. A system access control list (SACL) that controls how the system audits attempt to access the object.

An ACL contains a list of access control entries (ACEs). Each ACE specifies a set of access rights and contains a security identifier that identifies a trustee for whom the rights are allowed, denied, or audited. A trustee can be a user account, group account, or logon session. A logon session begins whenever a user logs on to a computer. All processes in a logon session have the same primary access token. The access token contains information about the security context of the logon session, including the user's SID, the logon identifier, and the logon SID. Keep in mind that the user is nothing because user's credential such as his/her username and password was created in the system. User's credential just another Windows object.

---

### **11.8.3 ACCESS TOKENS**

---

An access token is an object that describes the security context of a process or thread. The information in a token includes the identity and privileges

of the user account associated with the process or thread. When a user logs on, the system verifies the user's password by comparing it with information stored in a security database. If the password is authenticated, the system produces an access token. Every process executed on behalf of this user has a copy of this access token.

The system uses an access token to identify user when a thread interacts with a securable object or tries to perform a system task that requires privileges. Access tokens contain the following information:

- The SID for the user's account.
- SIDs for the groups of which the user is a member.
- A logon SID that identifies the current logon session.
- A list of the privileges held by either the user or the user's groups.
- An owner SID.
- The SID for the primary group.
- The default DACL that the system uses when the user creates a securable object without specifying a security descriptor.
- The source of the access token.
- Whether the token is a primary or impersonation token.
- An optional list of restricting SIDs.
- Current impersonation levels.
- Other statistics.

A primary token is an access token that is typically created only by the Windows kernel. It may be assigned to a process to represent the default security information for that process. The impersonation token is an access token that has been created to capture the security information of a client process, allowing a server to "impersonate" the client process in security operations.

Every process has a primary token that describes the security context of the user account associated with the process. By default, the system uses the primary token when a thread of the process interacts with a securable object. Moreover, a thread can impersonate a client account. Impersonation allows the thread to interact with securable objects using the client's security context.

A thread that is impersonating a client has both a primary token and an impersonation token. You can use the `OpenProcessToken()` function to retrieve a handle to the primary token of a process. Use the `OpenThreadToken()` function to retrieve a handle to the impersonation token of a thread. You can use the functions shown in Table 11.1 to manipulate access tokens.

**Table 11.1 Functions to manipulate access tokens**

Function	Description
AdjustTokenGroups()	Changes the group information in an access token.
AdjustTokenPrivileges()	Enables or disables the privileges in an access token. It does not grant new privileges or revoke existing ones.
CheckTokenMembership()	Determines whether a specified SID is enabled in a specified access token.
CreateRestrictedToken()	Creates a new token that is a restricted version of an existing token. The restricted token can have disabled SIDs, deleted privileges, and a list of restricted SIDs.
DuplicateToken()	Creates a new impersonation token that duplicates an existing token.
DuplicateTokenEx()	Creates a new primary token or impersonation token that duplicates an existing token.
GetTokenInformation()	Retrieves information about a token.
IsTokenRestricted()	Determines whether a token has a list of restricting SIDs.
OpenProcessToken()	Retrieves a handle to the primary access token for a process.
OpenThreadToken()	Retrieves a handle to the impersonation access token for a thread.
SetThreadToken()	Assigns or removes an impersonation token for a thread.
SetTokenInformation()	Changes a token's owner, primary group, or default DACL.

The access token functions use the following structures (Table 11.2) to describe the components of an access token.

**Table 11.2 Structures to describe the components**

Structure	Description
TOKEN_CONTROL	Information that identifies an access token.
TOKEN_DEFAULT_DACL	The default DACL that the system uses in the security descriptors of new objects created by a thread.
TOKEN_GROUPS	Specifies the SIDs and attributes of the group SIDs in an access token.
TOKEN_OWNER	The default owner SID for the security descriptors of new objects.
TOKEN_PRIMARY_GROUP	The default primary group SID for the security descriptors of new objects.
TOKEN_PRIVILEGES	The privileges associated with an access token. Also determines whether the privileges are enabled.
TOKEN_SOURCE	The source of an access token.
TOKEN_STATISTICS	Statistics associated with an access token.
TOKEN_USER	The SID of the user associated with an access token.

The access token functions use the following enumeration types (Table 11.3).

**Table 11.3 Enumeration Types**

Enumeration type	Specifies
TOKEN_INFORMATION_CLASS	Identifies the type of information being set or retrieved from an access token.
TOKEN_TYPE	Identifies an access token as a primary or impersonation token.

---

### **11.8.3 ACCESS RIGHTS FOR ACCESS-TOKEN OBJECTS**

---

An application cannot change the access control list of an object unless the application has the rights to do so. These rights are controlled by a security descriptor in the access token for the object. To get or set the security descriptor for an access token, call the `GetKernelObjectSecurity()` and

SetKernelObjectSecurity() functions. When you call the OpenProcessToken() or OpenThreadToken() function to get a handle to an access token, the system checks the requested access rights against the DACL in the token's security descriptor. The following are valid access rights for access-token objects:

- The DELETE, READ\_CONTROL, WRITE\_DAC, and WRITE\_OWNER standard access rights. Access tokens do not support the SYNCHRONIZE standard access right.
- The ACCESS\_SYSTEM\_SECURITY right to get or set the SACL in the object's security descriptor.
- The specific access rights for access tokens, which are listed in Table 11.4.

**Table 11.4 Access rights for access tokens**

Value	Meaning
TOKEN_ADJUST_DEFAULT	Required to change the default owner, primary group, or DACL of an access token.
TOKEN_ADJUST_GROUPS	Required to adjust the attributes of the groups in an access token.
TOKEN_ADJUST_PRIVILEGES	Required to enable or disable the privileges in an access token.
TOKEN_ADJUST_SESSIONID	Required to adjust the session ID of an access token. The SE_TCB_NAME privilege is required.
TOKEN_ASSIGN_PRIMARY	Required to attach a primary token to a process. The SE_ASSIGNPRIMARYTOKEN_NAME privilege is also required to accomplish this task.
TOKEN_DUPLICATE	Required to duplicate an access token.
TOKEN_EXECUTE	Combines STANDARD_RIGHTS_EXECUTE and TOKEN_IMPERSONATE.
TOKEN_IMPERSONATE	Required to attach an impersonation access token to a process.
TOKEN_QUERY	Required to query an access token.
TOKEN_QUERY_SOURCE	Required to query the source of an access token.
TOKEN_READ	Combines STANDARD_RIGHTS_READ and TOKEN_QUERY.



TOKEN_WRITE	Combines STANDARD_RIGHTS_WRITE, TOKEN_ADJUST_PRIVILEGES, TOKEN_ADJUST_GROUPS, and TOKEN_ADJUST_DEFAULT.
TOKEN_ALL_ACCESS	Combines all possible access rights for a token

---

## 11.9 SECURITY DESCRIPTORS

---

A security descriptor contains the security information associated with a securable object. A security descriptor consists of a SECURITY\_DESCRIPTOR structure and its associated security information. A security descriptor can include the following security information:

- SIDs for the owner and primary group of an object.
- A DACL that specifies the access rights allowed or denied to particular users or groups.
- A SACL that specifies the types of access attempts that generate audit records for the object.
- A set of control bits that qualify the meaning of a security descriptor or its individual members.

The Windows API provides functions for setting and retrieving the security information in an object's security descriptor. In addition, there are functions for creating and initializing a security descriptor for a new object. Applications working with security descriptors on Active Directory objects can use the Windows security functions or the security interfaces provided by the Active Directory Service Interfaces (ADSI).

---

### 11.9.1 SECURABLE OBJECTS

---

A securable object is an object that can have a security descriptor. All named Windows objects are securable. Some unnamed objects, such as process and thread objects, can have security descriptors too. For most securable objects, you can specify an object's security descriptor in the function call that creates the object. For example, you can specify a security descriptor in the CreateFile and CreateProcess functions.

In addition, the Windows security functions enable you to get and set the security information for securable objects created on operating systems other than Windows. The Windows security functions also provide support for using security descriptors with private, application-defined objects.

The following Table 11.5 shows the functions to use to manipulate the security information for some common securable objects.

**Table 11.5 Some common securable objects**

Object type	Security descriptor functions
Files or directories on an NTFS file system	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Named pipes Anonymous pipes	GetSecurityInfo, SetSecurityInfo
Processes Threads	GetSecurityInfo, SetSecurityInfo
File-mapping objects	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Access tokens	SetKernelObjectSecurity, GetKernelObjectSecurity
Window-management objects (window stations and desktops)	GetSecurityInfo, SetSecurityInfo
Registry keys	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Windows services	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Local or remote printers	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Network shares	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Interprocess synchronization objects (events, mutexes, semaphores, and waitable timers)	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo

Job objects	GetNamedSecurityInfo, SetNamedSecurityInfo, GetSecurityInfo, SetSecurityInfo
Directory service objects	These objects are handled by Active Directory Objects. For more information, see Active Directory Service Interfaces.

### **CHECK YOUR PROGRESS**

- Define the term Windows object.
- Write three access rights and their description for access tokens.
- What do you understand by security descriptor?

---

## **11.10 SUMMARY**

---

Winsock is a standard application programming interface (API) that allows two or more applications (or processes) to communicate either on the same machine or across a network and is primarily designed to foster data communication over a network. It is important to understand that Winsock is a network programming interface and not a protocol. The MFC supports programming with the Windows Sockets API by supplying two classes. One of these classes, CSocket, provides a high level of abstraction to simplify the network communications programming.

In Internet Protocol (IP), connection-oriented communication is accomplished through the TCP/IP protocol. TCP provides reliable error-free data transmission between two computers. When applications communicate using TCP, a virtual connection is established between the source computer and the destination computer. Connectionless communication behaves differently than connection-oriented communication, so the method for sending and receiving data is substantially different. In IP, connectionless communication is accomplished through UDP/IP.

The Winsock Catalog is a database that contains the different protocols available on the system. Winsock provides a method for determining which protocols are installed on a given workstation and returning a variety of characteristics for each protocol. If a protocol is capable of multiple behaviors, each distinct behavior type has its own catalog entry within the system. The CWnd object is a C++ window object, distinct from the HWND that represents a Windows window but containing it. CWnd can be used to derive child window classes, or use one of the many MFC classes derived from CWnd. Class CWnd is the base class for all windows, including frame windows, dialog boxes, child windows, controls, and control bars such as toolbars.

Access control refers to security features that control who can access

which resources in the operating system. Applications call access control functions to set who can access specific resources or control access to resources provided by the applications. Access Control is one of the important and fundamental topics in Windows SDK Platform from Security point of view. A security descriptor contains the security information associated with a securable object. A security descriptor consists of a SECURITY\_DESCRIPTOR structure and its associated security information.

---

## 11.11 TERMINAL QUESTIONS

---

1. What do you understand by Winsock? Explain its architecture.
2. Discuss about the windows socket and its types.
3. Describe the functions getpeername and getsockname.
4. Write a short note on Winsock Catalog.
5. Discuss about windows objects.
6. Give the meaning of access tokens. Explain about the information it contains.
7. What do you understand by security descriptors? Explain briefly.
8. Explain some common securable objects.

---

# UNIT-12 ADVANCE TOPICS AND CASE STUDY

---

## Structure

- 12.0 Introduction
- 12.1 Objectives
- 12.2 Active X Control
- 12.3 Component Object Model (COM)
- 12.4 COM+
- 12.5 Distributed Component Object Model (DCOM)
- 12.6 Application using Visual Basic
- 12.7 Example – Customer Database Input Screen
- 12.8 Summary
- 12.9 Terminal Questions

---

## 12.0 INTRODUCTION

---

This unit basically describes about the ActiveX, ActiveX control, COM (Component Object Model), enhanced version of COM ie COM+, DCOM (Distributed Component Object Model), and finally a project based on a case study which has been developed in Visual Basic programming. The ActiveX controls are based on the Active Template Library (ATL) with no external dependencies on third-party libraries such as the Visual C++ runtime. They can be used with Visual Basic 6.0, Visual C++, PowerBuilder and any other language that supports the Component Object Model (COM) and ActiveX standard. This also includes Visual Basic for Applications (VBA) in Microsoft Office, and scripting languages such as VBScript. Moreover, the ability to create a full-fledged ActiveX control using VB is an extremely important step for VB developers. Furthermore, a COM+ application is the primary unit of administration and security for Component Services and consists of a group of COM components that generally perform related functions.

On the other hand, DCOM provides the ability to use and reuse components dynamically, without recompiling, on any platform, from any language, at any time. This is perhaps the single most important aspect of DCOM. It also has a highly optimized marshaler for a set of common parameter types (ole automation), which gives DCOM advantage over CORBA, whose marshaler is more general and therefore slower. It allows for static and dynamic invocation of objects, multithreading, callback events, object locator, remote server activation,

security, and persistence. And most of all, it is already highly deployed in all Windows platforms since DCOM is really COM *'with a longer wire'*. Lastly, a brief overview about the application development using a small project in Visual Basic programming has been described which would be helpful for the project building.

---

## 12.1 OBJECTIVES

---

At the end of this unit you will come to know about the following:

- ActiveX control
- Component Object Model (COM)
- COM+
- Distributed Component Object Model (DCOM)
- Difference between the COM and DCOM
- Characteristics of COM, COM+, DCOM
- Issues/limitations of COM, COM+, DCOM
- Design and development of an application using Visual Basic

---

## 12.2 ACTIVEX CONTROL

---

ActiveX technology was introduced in Visual Basic 5.0 for the first time. ActiveX is the name for programmable elements, formerly known as OLE controls (OCXs) or OLE custom controls. The ActiveX programming specification is an extension of Microsoft Windows and the Object Linking and Embedding API. Microsoft uses the term ActiveX to describe a number of its COM (Component Object Model) technologies. However, when most people say "ActiveX", they are really referring to ActiveX controls which are nowadays used in most of the Internet programs; Microsoft's answer to Java applets. Earlier, ActiveX, was basically used for the windows like Win3.1, WinNT, Win2000; moreover, it was wrapped around the OLE and COM. Furthermore, like applets, programs that use ActiveX controls run on the client computers, not the server.

Microsoft first introduced the term ActiveX at the Internet Professional Developers Conference in March of 1996. ActiveX referred to the conference slogan "Activate the Internet" and was more a call-to-arms than a technology or architecture for developing applications. In a very short time, it became Microsoft's answer to the Java technology from Sun Microsystems. An ActiveX control is roughly equivalent to a Java applet. The difference is that while ActiveX controls can interface with Microsoft Windows better than Java, they only offer limited cross-platform support. Currently, ActiveX controls run in Windows and in Macintosh.

ActiveX controls can be created using a variety of languages or development tools, including C++, Visual Basic, PowerBuilder, or with scripting tools such as VBScript. ActiveX controls are small program building blocks that can be used to create distributed applications that work over the Internet through web browsers. Examples include customized applications for gathering data, viewing certain kinds of files, and displaying animation.

The ActiveX programming specification is an extension of Microsoft Windows and the Object Linking and Embedding (OLE) API. ActiveX applications are used mainly with Microsoft's Internet Explorer web browser. It is to be noted here that ActiveX and ActiveX controls are similar in that they are both designed to be downloaded and executed by web browsers. The difference is that while ActiveX controls can interface with Microsoft Windows more effectively than Java, they offer very little cross-platform support.

---

## **12.2.1 ACTIVEX COMPONENTS**

---

The components that you create using ActiveX technology are of different types. The following are the different types of ActiveX components.

---

### **12.2.1.1 ACTIVEX APPLICATIONS (ACTIVEX .EXE)**

---

An ActiveX application is a standalone application, such as MS-Word, MS-Excel etc. These applications provide objects that you can access and manipulate programmatically from an application written in Visual Basic or any application development tool that supports ActiveX.

***Note:-** Each component has a collection of properties, methods and events that can be accessed from outside. These properties, methods and events are collectively called as interface of the component.*

---

### **12.2.1.2 ACTIVEX CODE COMPONENTS (ACTIVEX .DLL)**

---

ActiveX code component is a collection (library) of programmable objects, where objects are generally related to a specific topic such as, financial functions, date and time functions etc. ActiveX code components do not run as separate applications, instead they are run in the same process area as the client (the application that is using the code component).

---

### **12.2.1.3 ACTIVEX CONTROLS (.OCX CONTROLS)**

---

ActiveX controls are user-defined controls. Formally they were called as OLE controls. Each ActiveX control does a specific job. For example, an ActiveX control may deal with displaying a calendar, another may deal with displaying running digital clock and so on. There are hundreds of ActiveX controls available from various vendors, whose primary job is creating ActiveX controls and sell them to developers.

---

### **12.2.1.4 ACTIVEX DOCUMENT (.VBD DOCUMENT)**

---

ActiveX documents are Internet pages. You can use ActiveX documents to create interactive Internet application. Each ActiveX document is a Web page. An ActiveX document can host ActiveX controls and can invoke the dialog boxes and so on. Visual Basic 6.0 has introduced DHTML application. DHTML application provides better alternative to ActiveX document application.

---

## 12.2.2 ACTIVEX CONTROL IN VISUAL BASIC

---

An ActiveX control is a component that may be added to the Form, like the controls in ToolBox. Three different types of ActiveX control in Visual Basic can be made. You can build an ActiveX Control without using any existing controls, designing your control completely from scratch. Moreover, you can *build a control that takes an existing control and extends its functionality. For example—* validating data entered into a TextBox. Furthermore, you can build an ActiveX control that is made up of existing controls (Constituent Controls). *For example—* grouping a Label and a TextBox to make a control that provides text input with a read-only prompt.

---

## 12.2.3 CREATING YOUR FIRST ACTIVEX CONTROL

---

The Visual Basic 6.0 Control Creation Edition makes creating ActiveX controls as easy as creating typical Visual Basic applications. If you have never seen just how easy it can be, you should definitely read on. This document is designed to give you a fast track overview of the simple process involved in creating ActiveX controls with Visual Basic.

If you follow the steps given below, and when you reach the end you will have built what is commonly called a "spinner" control. A *spinner control* is a graphical ActiveX control that allows the user to increment or decrement a value using a mouse instead of using a keyboard, as shown in the Fig. 12.1.



**Fig. 12.1 First ActiveX Control**

### Step 1: Create A Test Container

Start the Control Creation Edition, highlight Standard EXE, and click Open as shown below (Fig. 12.2). This creates the host application. This host will be used as the test container for the spinner control that is about to be created below.

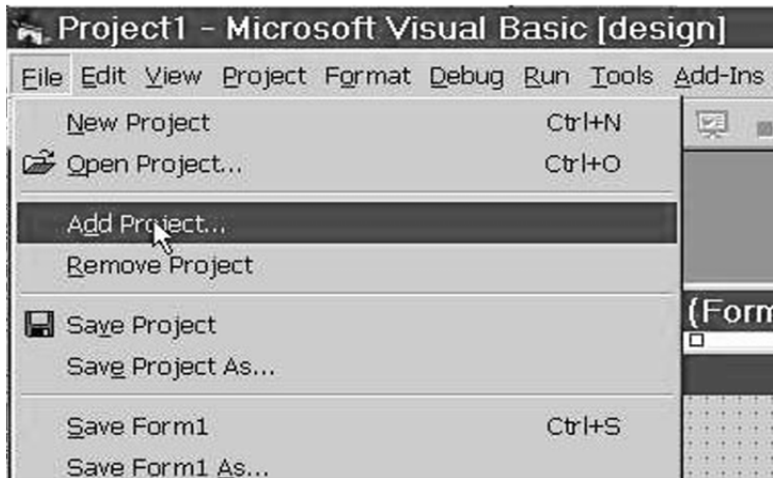


**Fig. 12.2 Creating A Test Container**



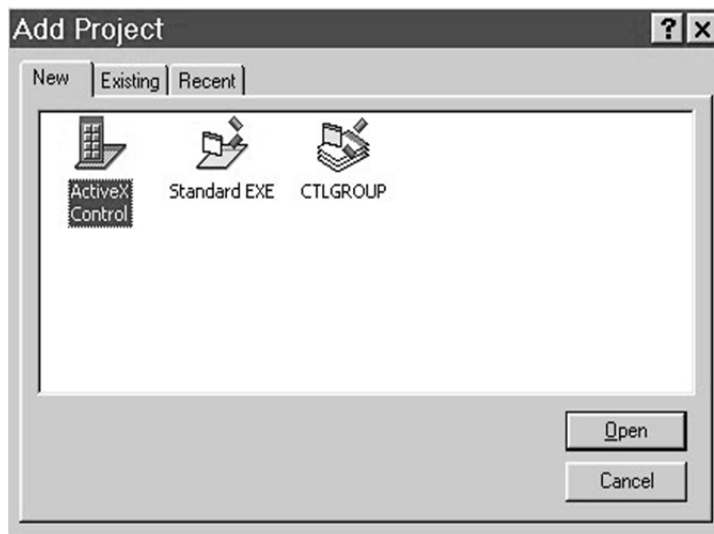
## Step 2: Add A Blank ActiveX Control Project

From the File Menu, select the Add Project Command (Fig. 12.3)



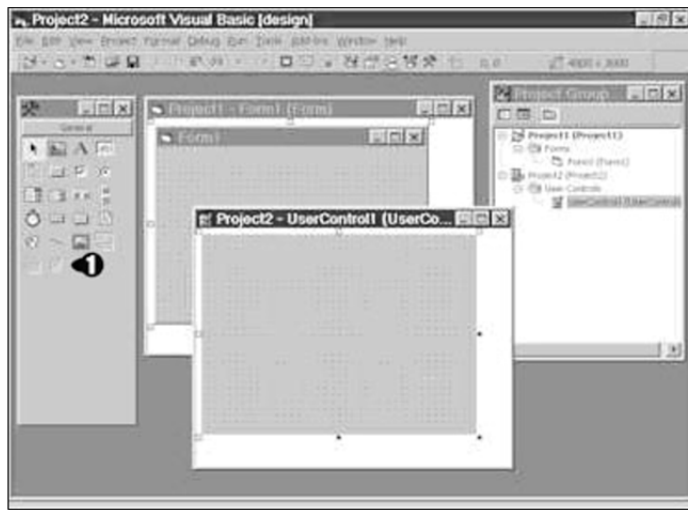
**Fig. 12.3 Adding A Blank ActiveX Control**

In the Add Project dialog, highlight ActiveX Control and Click Open (Fig. 12.4).



**Fig. 12.4 Adding Project**

At this point there should be two projects open. As you can see in the following diagram, both projects look extremely similar. Also note that there is a new control that is now visible in the toolbox (Fig 12.5). If you hover your mouse over this control in the Toolbox, the Tooltip should popup and display the current name of the control, "UserControl1". However, the icon in the Toolbox will be gray (disabled) at this point. Visual Basic uses the same visual metaphor for building ActiveX Controls as it does applications. Using this metaphor, you first "draw" the interface, set some properties, write some event driven code, and you are on your way.

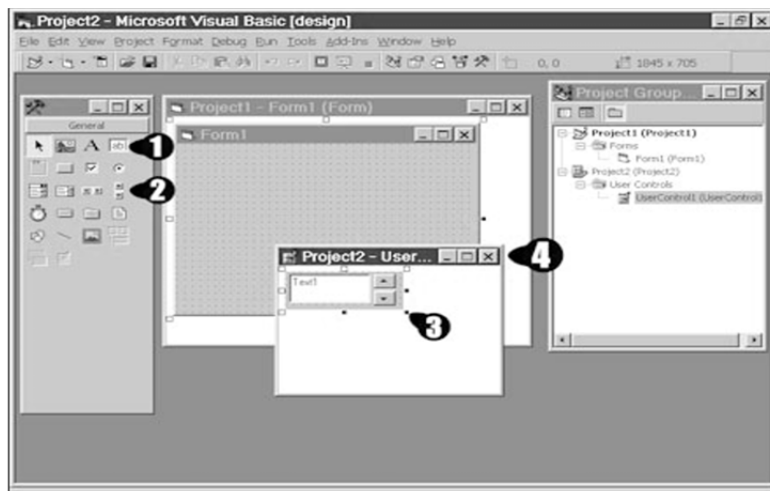


**Fig. 12.5 A new project IDE**

### Step 3: Draw The Visual Interface for The Control

Now, it is time to create the visual interface for the spinner control. The spinner control can be created using a powerful new feature of the Control Creation Edition - the ability to combine existing controls into new, more specialized controls. To create the spinner control, a standard textbox and a vertical scrollbar would be combined.

First, click on the textbox control in the Visual Basic toolbox. The textbox control is identified with the number 1 (Fig 12.6) below. Using the mouse, draw a small textbox in the upper left hand corner of the Project2 window. Second, click on the vertical scrollbar control in the toolbox and draw it just to the right of the textbox control. The vertical scrollbar control is identified with the number 2 (Fig 12.6) below. Finally, drag the control sizing handle to surround the newly drawn controls. The control sizing handle is identified with the number 3 in the Fig. 12.6 below. Your ActiveX spinner control should now look similar to the diagram below (Fig 12.6).



**Fig. 12.6 Labelling of controls**

## Step 4: Write Event Driven Code

At this point we now have a visual interface for a spinner control. The next step is to write some event driven code that will place the current value of the vertical scrollbar into the textbox. When the user clicks on the up or down arrow of the vertical scrollbar, the value displayed in the textbox needs to increment or decrement. To make this happen, some code needs to be written in the Change event of the vertical scrollbar. Double click on the vertical scrollbar that you just drew. This will display the code window. In the code window, type the following line of code:

```
text1.text = vscroll11.value
```

As soon as you type the "dot", Visual Basic displays a list of all allowable properties for the textbox. All ActiveX components contain this type of information and Visual Basic makes it automatically available when needed. After the code is entered, close the code window by clicking on the close box (#1 in Fig. 12.7). Finally, close the spinner control form by clicking on its close box (#2 in Fig 7 below).

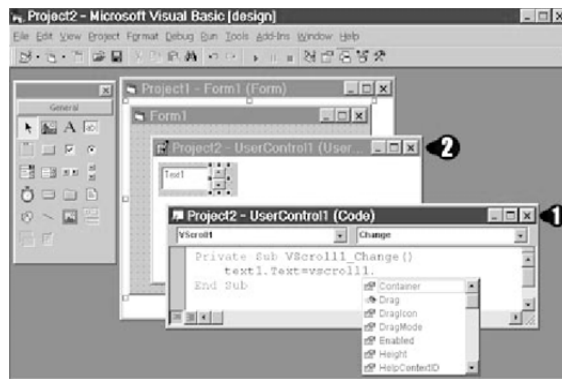


Fig. 12.7 Close box

## Step 5: Use and Test the Control

At this point, if all has gone well and you closed the spinner form, the spinner control will no longer appear gray in the toolbox and is ready to be used/tested (#1 in Fig 12.8). Your environment should now look similar to the Fig 12.8.

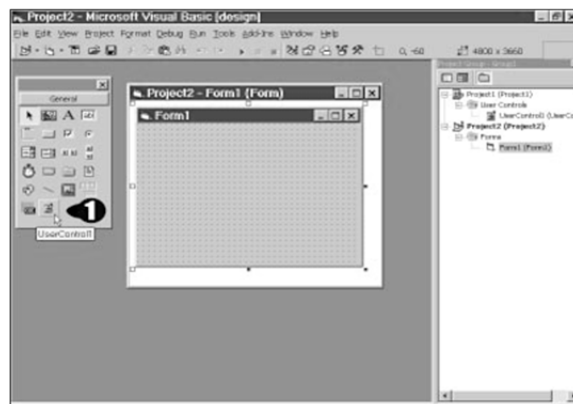


Fig. 12.8 Test the control

To test the newly created control, click on it in the toolbox and draw it on Form1 as shown below. Press F5 to run the application. As you click the up and down arrow in the spinner control, the value in the textbox changes, just as we coded it.

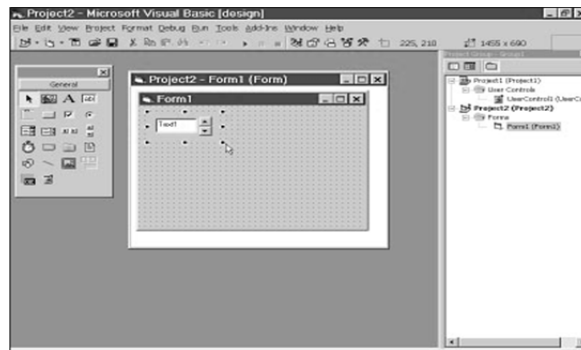


Fig. 12.9 Sinner control on Form1

---

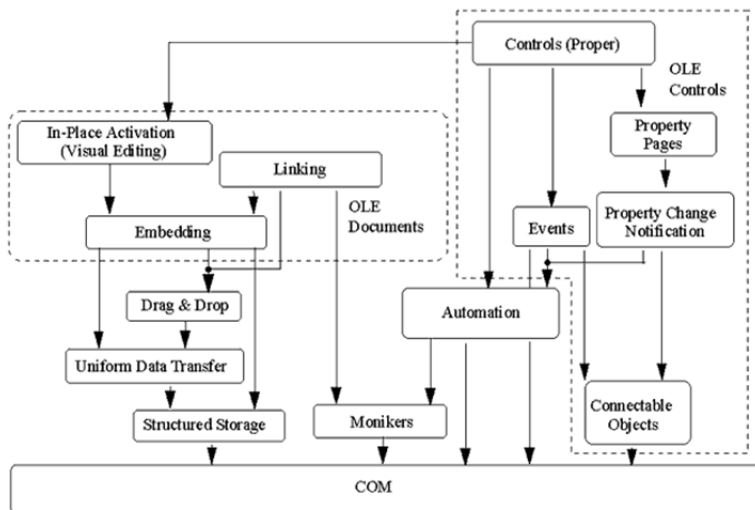
## 12.3 COMPONENT OBJECT MODEL

---

Basically, component object model or COM in short is the architecture on which ActiveX is based. COM is the software architecture that allows programmers to create components, which can be later used to build applications. The main theme of COM is its *reusability*. In other words, if one creates a component using COM specifications, it can be used in any language and any platform where COM is supported (Fig 12.10).

Moreover, COM is the underlying architecture that forms the foundation for higher-level software services, like those provided by OLE and ActiveX. The COM specifies the standards using which components are to be created. Microsoft is using COM to address specific areas like creating controls, servers etc. And the components created using COM specifications have the following characteristics:

- Defines a binary standard for component interoperability.
- They are platform independent.
- Provides for robust evolution of component-based applications and systems
- They are usable in Windows, Windows NT, Macintosh, and Unix.
- They are language independent. If one creates a component in Visual Basic, later can use that component in VC++, PowerBuilder or anywhere, where COM is supported.
- They are also extensible.
- COM is a general architecture for component software.
- Provides rich error and status reporting.
- Allows for shared memory management between components.
- Allows dynamic loading and unloading of components.



**Fig. 12.10 COM as the foundation for OLE technologies**

(Source: <https://www.cs.umd.edu/~pugh/com/>)

It is important to note that COM is a general architecture for component software. Although Microsoft is applying COM to address specific areas such as controls, compound documents, automation, data transfer, storage and naming, and others, any developer can take advantage of the structure and foundation that COM provides.

### 12.3.1 THE COMPONENT SOFTWARE PROBLEM

The most fundamental problem that COM solves is— “*How can a system be designed so that binary executables from different vendors, written in different parts of the world and at different times, are able to interoperate?*” To solve this problem, we must first find answers to these four questions:

- **Basic Interoperability-** How can developers create their own unique binary components, yet be assured that these binary components would interoperate with other binary components built by different developers?
- **Versioning-** How can one system component be upgraded without requiring all the system components to be upgraded?
- **Language Independence-** How can components written in different languages communicate?
- **Transparent Cross-Process Interoperability-** How can we give developers the flexibility to write components to run in-process or cross-process, and even cross-network, using one simple programming model?

Additionally, high performance is a requirement for a component software architecture. Although cross-process and cross-network transparency is a laudable goal, it is critical for the commercial success of a binary component marketplace that components interacting within the same address space be able to use each other's services without any undue "system" overhead. Otherwise, the components

will not realistically be scalable down to very small, lightweight pieces of software equivalent to C++ classes or graphical user interface (GUI) controls.

---

### 12.3.2 COM FUNDAMENTALS

---

The Component Object Model defines several fundamental concepts that provide the model's structural underpinnings. These include:

- A binary standard for function calling between components.
- A provision for strongly-typed groupings of functions into interfaces.
- A base interface providing.
- A way for components to dynamically discover the interfaces implemented by other components.
- Reference counting to allow components to track their own lifetime and delete themselves when appropriate.
- A mechanism to identify components and their interfaces uniquely, worldwide.
- A "component loader" to set up component interactions and, additionally (in the cross-process and cross-network cases), to help manage component interactions.

---

### 12.3.3 BINARY STANDARD

---

For any given platform (hardware and operating system combination), COM defines a standard way to lay out *virtual function tables (vtables)* in memory, and a standard way to call functions through the vtables. Thus, any language that can call functions via pointers (C, C++, Smalltalk, Ada, and even BASIC) all can be used to write components that can interoperate with other components written to the same binary standard. Indirection (the client holds a pointer to a vtable) allows for vtable sharing among multiple instances of the same object class. On a system with hundreds of object instances, vtable sharing can reduce memory requirements considerably, because additional vtables pointing to the same component instance consume much less memory than multiple instances of the same component.

#### **CHECK YOUR PROGRESS**

- Describe some ActiveX components.
- What do you understand by the COM?
- Write any three basic concepts of COM.

---

## 12.4 COM+

---

COM+ is an enhanced version of COM that provides better security and improved performance. Component Object Model+ (COM+) is a binary interoperability standard defined by Microsoft that specifies a model for distributed object communication. The COM+ defines communication by separating objects into clients and servers. The client is defined as an object that wants to access a particular service, while the server is an object that provides service. The client and server can communicate with each other independently of the programming language in which they are defined and independently of the operating system that lies between them.

The basic COM specification only established a distributed communication model between a client and a server without any performance optimizations. COM+ can be considered a successor to COM, with features related to Microsoft Transaction Server (MTS).

COM+ provides the following features:

- Enhanced security compared to COM with the help of access regulation.
- Support for application recycling.
- Support of partitions, where several COM+ versions can be installed simultaneously on the same machine.
- COM+ components can also provide services without components.
- COM+ applications possess external interfaces that provide a Web service interface for communication using XML.

To ensure reliability, COM+ uses a memory activation mechanism. With this mechanism, the amount of virtual memory is calculated prior to the creation of a server object. If less memory is available, the activation or creation of a COM+ object fails. Thus, COM+ components cannot suffer a software crash due to overload. The COM+ can be used to develop enterprise-wide, mission-critical, distributed applications for Windows.

Moreover, it is applicable; If you are a system administrator, you will be installing, deploying, and configuring COM+ applications and their components. If you are an application programmer, you will be writing components and integrating them as applications. If you are a tools vendor, you will be developing or modifying tools to work in the COM+ environment.

**Note:-** COM+ is designed primarily for Microsoft Visual C++ and Microsoft Visual Basic developers. COM+ version 1.5 is included in Windows starting with Windows XP and Windows Server 2003. COM+ version 1.0 is included in Windows 2000.

---

## 12.5 DISTRIBUTED COMPONENT OBJECT MODEL

---

Distributed Component Object Model (DCOM) is a proprietary Microsoft technology that allows Component Object Model (COM) software to communicate across a network. The DCOM is enhanced with COM applications to facilitate remote procedural calls (RPC) and a Distributed Computing

Environment (DCE) dedicated to Windows application and platform support. Initially, Microsoft developed COM to communicate with processes running on one machine. Microsoft created DCOM to allow distributed application and process communication with machines across an entire network.

Traditional COM components can only perform inter-process communication across process boundaries on the same machine. DCOM uses the RPC mechanism to transparently send and receive information between COM components (i.e., clients and servers) on the same network. DCOM was first made available in 1995 with the initial release of Windows NT 4. It serves the same purpose as IBM's DSOM protocol, which is the most popular implementation of CORBA. Unlike CORBA, which runs on many operating systems, DCOM is currently implemented only for Windows.

---

## **12.5.1 PROBLEMS SOLVED BY DCOM**

---

An extension of COM, DCOM solves a few inherent problems with the COM model to better use over a network.

---

### **12.5.1.1 MARSHALLING**

---

Marshalling solves a need to pass data from one COM object instance to another on a different computer – in programming terms, this is called “passing arguments.” For example, if I wanted Zaphod’s last name, I would call the COM Object LastName with the argument of Zaphod. The LastName function would use a Remote Procedure Call (RPC) to ask the other COM object on the target server for the return value for LastName(Zaphod), and then it would send the answer – Beeblebrox – back to the first COM object.

---

### **12.5.1.2 DISTRIBUTED GARBAGE COLLECTION**

---

Designed to scale DCOM in order to support high volume internet traffic, Distributed Garbage Collection also addresses a way to destroy and reclaim completed or abandoned DCOM objects to avoid blowing up the memory on web servers. In turn, it communicates with the other servers in the transaction chain to let them know they can get rid of the objects related to a transaction.

---

### **12.5.1.3 USING DCE/RPC AS THE UNDERLYING RPC MECHANISM**

---

To achieve the previous items and to attempt to scale to support high volume web traffic, Microsoft implemented DCE (Distributed Computing Environment)/RPC as the underlying technology for DCOM – which is where the D in DCOM came from.

---

## **12.5.2 WORKING OF DCOM**

---

In order for the DCOM to work, the COM object needs to be configured correctly on both computers – in our experience they rarely were, and you had to uninstall and reinstall the objects several times to get them to work.



The Windows Registry contains the DCOM configuration data in three identifiers:

- **CLSID**– The Class Identifier (CLSID) is a Global Unique Identifier (GUID). Windows stores a CLSID for each installed class in a program. When you need to run a class, you need the correct CLSID, so Windows knows where to go and find the program.
- **PROGID**– The Programmatic Identifier (PROGID) is an optional identifier a programmer can substitute for the more complicated and strict CLSID. PROGIDs are usually easier to read and understand. A basic PROGID for our previous example could be *Hitchiker.LastName*. There are no restrictions on how many PROGIDs can have the same name, which causes issues on occasion.
- **APPID**– The Application Identifier (APPID) identifies all of the classes that are part of the same executable and the permissions required to access it. DCOM cannot work if the APPID isn't correct. You will probably get permissions errors trying to create the remote object, in my experience.

A basic DCOM transaction looks like the following

- The client computer requests the remote computer to create an object by its CLSID or PROGID. If the client passes the APPID, the remote computer looks up the CLSID using the PROGID.
- The remote machine checks the APPID and verifies the client has permissions to create the object.
- DCOMLaunch.exe (if an exe) or DLLHOST.exe (if a dll) will create an instance of the class the client computer requested.
- Communication is successful!
- The Client can now access all functions in the class on the remote computer

One widely criticized aspect of the DCOM model however, is that there is no absolute way of addressing an object instance – everything is done through object interfaces. As such, it can be difficult to manage a large set of worker object instances or temporarily disconnect and reconnect at a later time.

Another problem DCOM is facing is that currently there is no good solution to the problem of keeping track of possibly thousands of objects spread over thousands of computers on the network. The user has to supply the network address of the host machine for the server object, or that address must be hard-coded in the client application itself.

Some also say that DCOM is hard to program but that depends entirely on the language being used and the level of support it offers for DCOM programming. Writing DCOM objects in VC++ can be pretty involving but allows for slimmer and faster implementations. Using VB5 or J++ for that purpose is really trivial and COM+ will soon make it trivial for VC++ as well.

---

### 12.5.3 DCOM VS. CORBA

---

Common Object Request Broker Architecture (CORBA) is a JAVA based application and functions basically the same as DCOM. Unlike DCOM, CORBA isn't tied to any particular OS, and works on UNIX, Linux, SUN, OS X, and other UNIX-based platforms. Neither proved secure or scalable enough to become a standard for high volume web traffic. DCOM and CORBA didn't play well with firewalls, so HTTP became the default standard protocol for the internet.

---

### 12.5.4 SIGNIFICANCE OF DCOM

---

DCOM didn't win the battle to become the standard protocol for the Internet, but it remains integrated into the Windows OS and is how many Windows services communicate – like Microsoft Management Console (MMC). Since DCOM can run programs on other computers, hackers can leverage it for lateral movement attacks through your network, gaining access to more data. This activity can be difficult to detect because it's not malware or hacker tools: all it takes to access DCOM is PowerShell.

#### **CHECK YOUR PROGRESS**

- Define the term COM+.
- Write any two features of COM+.
- What do you understand by DCOM?
- Explain the need of DCOM.

---

## 12.6 DESIGNING APPLICATION USING VISUAL BASIC

---

We've finished looking at most of the Visual Basic tools and been introduced to most of the Basic language features. Thus far, to run any of the applications studied, we needed Visual Basic. Now in this topic, we would learn the steps of developing a stand-alone application that can be run on any Windows-based machine. We'd also look at some new components that help make up applications. In this application we would see how a small project can be developed in Visual Basic.

---

### 12.6.1 DESIGNING AN APPLICATION

---

Before starting the actual process of application development for a good interface, setting the object properties, inserting the basic code for various controls, menus etc.; we need to look about many things ahead.

**Step 1:** First step is to know the processes and functions needed for the application after knowing the main problems or requirements of the organization or on which it is intended to be developed and performed accordingly. Moreover,

note down the inputs and outputs. For this a flowchart or framework of application's process is to be made.

**Step 2:** Decide what tools are needed for the application. Do the built-in Visual Basic tools and functions are sufficient for the application? Or some tools or functions are to be taken of your own?

**Step 3:** Third step is to make a good and simple interface so that the users could easily interact with the application. Some good images can also be used for good looking but should not be much. Make the interface consistent with other Windows applications. Familiarity is always good in program design.

**Step 4:** Make the code readable and traceable for the functions, procedures, inputs and outputs. Code must be very simple but robust; and it must cover all the inputs and outputs. Moreover, the important thing is easy to debug. It must also be understood easily by other programmers whenever needed. It should be reusable so that it could be used in future too and save time. Make your code 'user-friendly.' Try to anticipate all possible ways a user can mess up in using your application. It's fairly easy to write an application that works properly when the user does everything correctly. It's difficult to write an application that can handle all the possible wrong things a user can do.

**Step 5:** Last step is the debugging of code. It can be done manually if application is small or having less lines of code or can be done using tools if lines of code is larger. It is better to test the application by giving or letting other people to use. Other people input the values as per their choice and this really works good in finding bugs of code. Other testing methods like black box testing and white box testing can also be used.

---

## 12.6.2 USING GENERAL SUB PROCEDURES IN APPLICATIONS

---

So far in this class, the only procedures we have studied are the event-driven procedures associated with the various tools. Most applications have tasks not related to objects that require some code to perform these tasks. Such tasks are usually coded in a general Sub procedure (essentially the same as a subroutine in other languages).

Using general Sub procedures can help in dividing a complex application into more manageable units of code. This helps meet the above stated goals of readability and reusability.

**Defining a Sub Procedure:** The form for a general Sub procedure named *GenSubProc* is:

```
Sub GenSubProc(Arguments) 'Definition header
End Sub
```

The definition header names the Sub procedure and defines any arguments passed to the procedure. Arguments are a comma-delimited list of variables passed to and/or from the procedure. If there are arguments, they must be declared and typed in the definition header in this form:

```
Var1 As Type1, Var2 As Type2, ...
```

**Example:** Here is a Sub procedure (USMexConvert) that accepts as inputs an amount in US dollars (USDollars) and an exchange rate (UStoPeso). It then outputs an amount in Mexican pesos (MexPesos).

```
Sub USMexConvert (USDollars As Single,  
UStoPeso As Single, MexPesos As Single)  
  
MexPesos = UsDollars * UsToPeso  
  
End Sub
```

---

### 12.6.2.1 CALLING A SUB PROCEDURE

---

There are two ways to call or invoke a Sub procedure. You can also use these to call event-driven procedures.

*Method-1:* Call GenlSubProc(Arguments) (if there are no Arguments, do not type the parentheses)

*Method-2:* GenlSubProc Arguments

Method-1 is normally preferred — it's more consistent with calling protocols in other languages and it cleanly delineates the argument list. It seems most Visual Basic programmers use Method-2, though. I guess they hate typing parentheses! Choose the method you feel more comfortable with.

**Example:** *To call our dollar exchange routine, we could use:*

```
Call USMexConvert (USDollars, UStoMex, MexPesos) or  
USMexConvert USDollars, UStoMex, MexPesos
```

---

### 12.6.2.2 LOCATING GENERAL SUB PROCEDURES

---

General Sub procedures can be located in one of two places in your application: attached to a form or attached to a module. Place the procedure in the form if it has a purpose specifically related to the form. Place it in a module if it is a general purpose procedure that might be used by another form or module or another application.

Whether placing the procedure in a form or module, the methods of creating the procedure are the same. Select or open the form or module's code window. Make sure the window's Object list says (General) and the Procedure list says (Declarations). You can now create the procedure by selecting Add Procedure from Visual Basic's Tools menu. A window appears allowing you to select Type Sub and enter a name for your procedure.

Another way to create a Sub is to go to the last line of the General Declarations section, type Sub followed by a space and the name of your procedure, then, hit Enter. With either method for establishing a Sub, Visual Basic will form a template for your procedure. Fill in the argument list and write your Basic code. In selecting the Insert Procedure menu item, note another option for

your procedure is Scope. You have the choice of Public or Private. The scope word appears before the Sub word in the definition heading. If a module procedure is Public, it can be called from any other procedure in any other module. If a module procedure is Private, it can only be called from the module it is defined in. Note, scope only applies to procedures in modules. By default, all event procedures and general procedures in a form are Private - they can only be called from within the form. You must decide the scope of your procedures.

---

### 12.6.2.3 PASSING ARGUMENTS TO SUB PROCEDURES

---

A quick word on passing arguments to procedures. By default, they are passed by reference. This means if an argument is changed within the procedure, it will remain changed once the procedure is exited. C programmers have experienced the concept of passing by value, where a parameter changed in a procedure will retain the value it had prior to calling the routine. Visual Basic also allows calling by value. To do this, place the word ByVal in front of each such variable in the Argument list.

---

### 12.6.2.4 USING GENERAL FUNCTION PROCEDURES IN APPLICATIONS

---

Related to Sub procedures are Function procedures. A Function procedure, or simply *Function*, performs a specific task within a Visual Basic program and returns a value. We've seen some built-in functions such as the MsgBox and the Format function.

*Defining a Function:* The form for a general Function named GenlFcn is given as:

```
Function GenlFcn(Arguments) As Type 'Definition header
.
.
GenlFcn = ...
End Function
```

The definition header names the Function and specifies its Type (the type of the returned value) and defines any input Arguments passed to the function. Note that somewhere in the function, a value for GenlFcn must be computed for return to the calling procedure.

*Function Example:* Here is a Function named CylVol that computes the volume of a cylinder of known height (Height) and radius (Radius).

```
Function CylVol(Height As Single, Radius As Single) As Single
Dim Area As Single
Const PI = 3.1415926
Area = PI * Radius ^ 2
CylVol = Area * Height
```

```
End Sub
```

*Calling a Function:* To call or use a Function, you equate a variable (of proper type) to the Function, with its arguments. That is, if the Function GenlFunc is of Type Integer, then use the code segment:

```
Dim RValue as Integer  
. .  
RValue = GenlFunc(Arguments)
```

**Example:** To call the volume computation function, we could use:

```
Dim Volume As Single  
. .  
Volume = CylVol(Height, Radius)
```

*Locating Function Procedures:* Like Sub procedures, Functions can be located in forms or modules. They are created using exactly the same process described for Sub procedures, the only difference being you use the keyword Function. And, like Sub procedures, Functions (in modules) can be Public or Private.

---

### 12.6.3 ADDING MENUS TO AN APPLICATION

---

As mentioned earlier, it is important that the interface of the application be familiar to a seasoned, or not-so-seasoned, Windows user. One such familiar application component is the Menu bar. Menus are used to provide a user with choices that control the application. Menus are easily incorporated into Visual Basic programs using the Menu Editor. A good way to think about elements of a menu structure is to consider them as a hierarchical list of command buttons that only appear when pulled down from the menu bar. When you click on a menu item, some action is taken. Like command buttons, menu items are named, have captions, and have properties.

**Example:** Here is a typical menu structure:

<i>File</i>	<i>Edit</i>	<i>Format</i>
New	Cut	Bold
Open	Copy	Italic
Save	Paste	Underline
		Size
		10
		16
		20

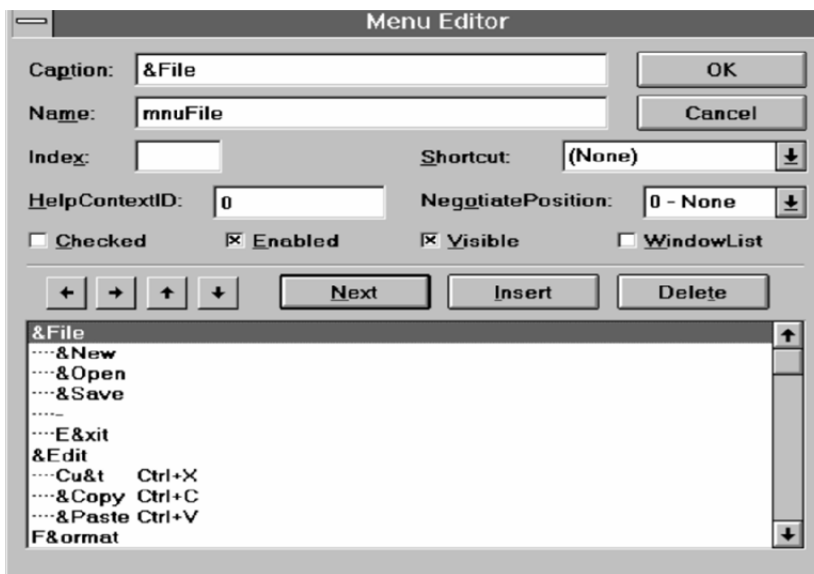
The level of indentation indicates position of a menu item within the hierarchy. For example, New is a sub-element of the File menu. The line under Save in the File menu is a separator bar (separates menu items).

With this structure, the Menu bar would display:

**File                      Edit                      Format**

The sub-menus appear when one of these ‘top’ level menu items is selected. Note the Size sub-menu under Format has another level of hierarchy. It is good practice to not use more than two levels in menus. Each menu element will have a Click event associated with it. The Menu Editor allows us to define the menu structure, adding access keys and shortcut keys, if desired. We then add code to the Click events we need to respond to.

The Menu Editor is selected from the Tools menu bar or by clicking the Menu Editor on the toolbar. This selection can only be made when the form needing the menu is active. Upon selecting the editor, and entering the example menu structure, the editor window looks like below in Fig. 12.13.



**Fig. 12.13 Menu editor**

The Caption box is where you type the text that appears in the menu bar. Access keys are defined in the standard way using the ampersand (&). Separator bars (a horizontal line used to separate menu items) are defined by using a Caption of a single hyphen (-). When assigning captions and access keys, try to use conform to any established Windows standards.

The Name box is where you enter a control name for each menu item. This is analogous to the Name property of command buttons and is the name used to set properties and establish the Click event procedure for each menu item. Each menu item must have a name, even separator bars! The prefix mnu is used to name menu items. Sub-menu item names usually refer back to main menu headings. For example, if the menu item New is under the main heading File menu, use the name mnuFileNew.

The Index box is used for indexing any menu items defined as control arrays. The Shortcut dropdown box is used to assign shortcut keystrokes to any item in a menu structure. The shortcut keystroke will appear to the right of the caption for the menu item. An example of such a keystroke is using Ctrl+X to cut text. The HelpContextID and NegotiatePosition boxes relate to using on-line help and object linking embedding, and are beyond the scope of this discussion. Each menu item has four properties associated with it. These properties can be set at design time using the Menu Editor or at run-time using the standard dot notation. These properties are given in Table 12.1.

**Table 12.1 Properties of Menu Editor**

Property	Description
Checked	Used to indicate whether a toggle option is turned on or off. If True, a check mark appears next to the menu item.
Enabled	If True, menu item can be selected. If False, menu item is grayed and cannot be selected.
Visible	Controls whether the menu item appears in the structure.
WindowList	WindowList

At the bottom of the Menu Editor form is a list box displaying the hierarchical list of menu items. Sub-menu items are indented to their level in the hierarchy. The right and left arrows adjust the levels of menu items, while the up and down arrows move items within the same level. The Next, Insert, and Delete buttons are used to move the selection down one line, insert a line above the current selection, or delete the current selection, respectively.

---

## 12.7 EXAMPLE - CUSTOMER DATABASE INPUT SCREEN

---

Now, let us discuss an application intended with the sports store for the entry of customer database containing some fields. A new sports store wants you to develop an input screen for its customer database as shown in Fig. 12.15. The required input information is:

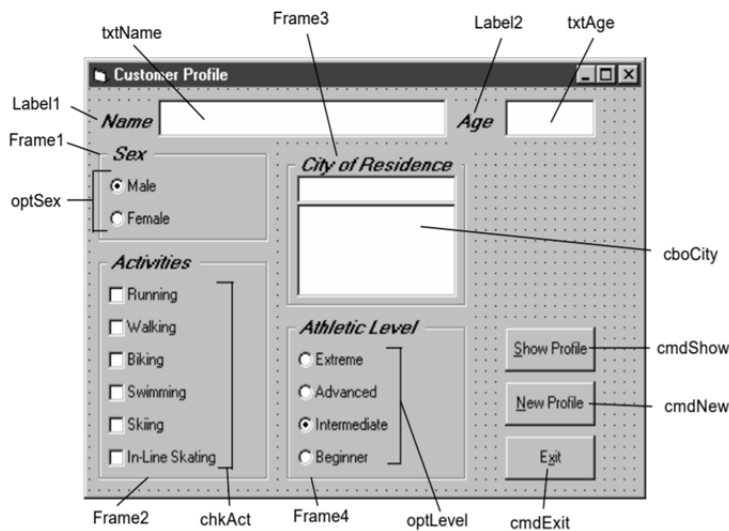
1. Name
2. Age
3. City of Residence
4. Sex (Male or Female)
5. Activities (Running, Walking, Biking, Swimming, Skiing and/or In-Line Skating)
6. Athletic Level (Extreme, Advanced, Intermediate, or Beginner)



Set up the screen so that only the Name and Age (use text boxes) and, perhaps, City (use a combo box) need to be typed; all other inputs should be set with check boxes and option buttons. When a screen of information is complete, display the summarized profile in a message box. This profile message box will look like shown in Fig. 12.14.



**Fig. 12.14 Profile message box**



**Fig. 12.15 Output of the program**

---

## 12.7.1 SETTING THE PROPERTIES

---

**Form** frmCustomer:

BorderStyle = 1 - Fixed Single

Caption = Customer Profile

**CommandButton** cmdExit:

Caption = Exit

**Frame** Frame3:

Caption = City of Residence  
FontName = MS Sans Serif  
FontBold = True  
FontSize = 9.75  
FontItalic = True

**ComboBox** cboCity:

Sorted = True  
Style = 1 - Simple Combo

**CommandButton** cmdNew:

Caption = New Profile

**CommandButton** cmdShow:

Caption = Show Profile

**Frame** Frame4:

Caption = Athletic Level  
FontName = MS Sans Serif  
FontBold = True  
FontSize = 9.75  
FontItalic = True

**OptionButton** optLevel:

Caption = Beginner Index = 3

**OptionButton** optLevel:

Caption = Intermediate Index = 2  
Value = True

**OptionButton** optLevel:

Caption = Advanced Index = 1

**OptionButton** optLevel:

Caption = Extreme  
Index = 0

**Frame** Frame1:

Caption = Sex  
FontName = MS Sans Serif  
FontBold = True  
FontSize = 9.75  
FontItalic = True

**OptionButton** optSex:

Caption = Female

Index = 1

**OptionButton** optSex:

Caption = Male

Index = 0

Value = True

**Frame** Frame2:

Caption = Activities

FontName = MS Sans Serif

FontBold = True FontSize = 9.75

FontItalic = True

**CheckBox** chkAct:

Caption = In-Line Skating

Index = 5

**CheckBox** chkAct:

Caption = Skiing

Index = 4

**CheckBox** chkAct:

Caption = Swimming

Index = 3

**CheckBox** chkAct:

Caption = Biking

Index = 2

**CheckBox** chkAct:

Caption = Walking

Index = 1

**CheckBox** chkAct:

Caption = Running

Index = 0

**TextBox** txtName:

FontName = MS Sans Serif

FontSize = 12

**Label** Labell1:

Caption = Name

FontName = MS Sans Serif

FontBold = True

FontSize = 9.75

```
FontItalic = True
```

**TextBox txtAge:**

```
FontName = MS Sans Serif
```

```
FontSize = 12
```

**Label Label2:**

```
Caption = Age
```

```
FontName = MS Sans Serif
```

```
FontBold = True
```

```
FontSize = 9.75
```

```
FontItalic = True
```

---

## 12.7.2 CODE

---

**General Declarations:**

```
Option Explicit
```

```
Dim Activity As String
```

**cmdExit Click Event:**

```
Private Sub cmdExit_Click()
```

```
End End Sub
```

**cmdNew Click Event:**

```
Private Sub cmdNew_Click()
```

```
'Blank out name and reset check boxes
```

```
Dim I As Integer
```

```
txtName.Text = ""
```

```
txtAge.Text = ""
```

```
For I = 0 To 5
```

```
    chkAct(I).Value = vbUnchecked
```

```
Next I
```

```
End Sub
```

**cmdShow Click Event:**

```
Private Sub cmdShow_Click()
```

```
Dim NoAct As Integer, I As Integer
```

```
Dim Msg As String, Pronoun As String
```

```
'Check to make sure name entered
```

```
If txtName.Text = "" Then
```

```

        MsgBox "The profile requires a name.", vbOKOnly +
        vbCritical, "No Name Entered"
Exit Sub
End If

'Check to make sure age entered
If txtAge.Text = "" Then
    MsgBox "The profile requires an age.", vbOKOnly +
    vbCritical, "No Age Entered"
    Exit Sub
End If

'Put together customer profile message
Msg = txtName.Text + " is" + Str$(txtAge.Text) + "
years old."
+ vbCr

If optSex(0).Value = True Then
Pronoun = "He " Else Pronoun = "She "
Msg = Msg + Pronoun + "lives in " + cboCity.Text + "."
+ vbCr
Msg = Msg + Pronoun + "is a" If optLevel(3).Value =
False Then
Msg = Msg + "n " Else Msg = Msg + " " Msg = Msg +
Activity + "
level athlete." + vbCr NoAct = 0

For I = 0 To 5
    If chkAct(I).Value = vbChecked Then
        NoAct = NoAct + 1
Next I

If NoAct > 0 Then
Msg = Msg + "Activities include:" + vbCr
    For I = 0 To 5

```

```

        If chkAct(I).Value = vbChecked Then
            Msg = Msg + String$(10, 32) + chkAct(I).Caption +
vbCr
        Next I
    Else
        Msg = Msg + vbCr
    End If
        MsgBox Msg, vbOKOnly, "Customer Profile"
    End Sub

```

### **Form Load Event:**

```

Private Sub Form_Load()
    'Load combo box with potential city names
    cboCity.AddItem "Seattle"
    cboCity.Text = "Seattle"
    cboCity.AddItem "Bellevue"
    cboCity.AddItem "Kirkland"
    cboCity.AddItem "Everett"
    cboCity.AddItem "Mercer Island"
    cboCity.AddItem "Renton"
    cboCity.AddItem "Issaquah"
    cboCity.AddItem "Kent"
    cboCity.AddItem "Bothell"
    cboCity.AddItem "Tukwila"
    cboCity.AddItem "West Seattle"
    cboCity.AddItem "Edmonds"
    cboCity.AddItem "Tacoma"
    cboCity.AddItem "Federal Way"
    cboCity.AddItem "Burien"
    cboCity.AddItem "SeaTac"
    cboCity.AddItem "Woodinville"
    Activity = "intermediate"
End Sub

```

### **optLevel Click Event:**

```
Private Sub optLevel_Click(Index As Integer)
'Determine activity level Select Case Index
Case 0 Activity = "extreme"
Case 1 Activity = "advanced"
Case 2 Activity = "intermediate"
Case 3 Activity = "beginner"
End Select
End Sub
```

### **txtAge KeyPress Event:**

```
Private Sub
txtAge_KeyPress(KeyAscii As Integer)
'Only allow numbers for age
If (KeyAscii >= vbKey0 And KeyAscii <= vbKey9)
Or KeyAscii = vbKeyBack Then
Exit Sub
Else
KeyAscii = 0
End If
End Sub
```

---

## **12.8 SUMMARY**

---

ActiveX is the name for programmable elements, formerly known as OLE controls (OCXs) or OLE custom controls. The ActiveX programming specification is an extension of Microsoft Windows and the Object Linking and Embedding API. Microsoft uses the term ActiveX to describe a number of its COM (Component Object Model) technologies.

ActiveX controls can be created using a variety of languages or development tools, including C++, Visual Basic, PowerBuilder, or with scripting tools such as VBScript. Different types of ActiveX components are: ActiveX Applications (ActiveX .EXE), ActiveX Code Components (ActiveX .DLL), ActiveX Controls (.OCX controls), ActiveX Document (. VBD Document).

Component Object Model (COM) is the software architecture that allows

programmers to create components, which can be later used to build applications. The main theme of COM is its *reusability*. In other words, if one creates a component using COM specifications, it can be used in any language and any platform where COM is supported.

COM+ is an enhanced version of COM that provides better security and improved performance. Component Object Model+ (COM+) is a binary interoperability standard defined by Microsoft that specifies a model for distributed object communication. COM+ defines communication by separating objects into clients and servers.

Distributed Component Object Model (DCOM) is a proprietary Microsoft technology that allows COM software to communicate across a network. The DCOM is enhanced with COM applications to facilitate remote procedural calls (RPC) and a Distributed Computing Environment (DCE) dedicated to Windows application and platform support. Initially, Microsoft developed the COM to communicate with processes running on one machine. Microsoft created the DCOM to allow distributed application and process communication with machines across an entire network.

---

## 12.9 TERMINAL QUESTIONS

---

1. What do you understand by ActiveX controls? Explain its components.
2. Discuss about the creation of first ActiveX control.
3. What do you understand by COM? Explain its characteristics.
4. Write a short note on basic concepts of COM.
5. Give the meaning of COM+. Moreover, explain its main features.
6. What do you understand by DCOM? Explain the problems solved by it.
7. Discuss and Develop an application entitled “Student Enrollment System.” using Visual Basic.

### **BIBLIOGRAPH**

- [1]. [https://en.wikipedia.org/wiki/Windows\\_API](https://en.wikipedia.org/wiki/Windows_API)
- [2]. <https://www.computerhope.com/>
- [3]. Programming Windows with MFC, Second Edition, by Jeff Prosise
- [4]. [https://en.wikibooks.org/wiki/Windows\\_Programming](https://en.wikibooks.org/wiki/Windows_Programming)
- [5]. Programming Windows, 5<sup>th</sup> edition, by Charles Petzold