LOVELY PROFESSIONAL UNIVERSITY

WINDOWS PROGRAMMING

Edited By

Kumar Vishal

Printed by
EXCEL BOOKS PRIVATE LIMITED
A-45, Naraina, Phase-I,
New Delhi-110028
for
Lovely Professional University
Phagwara

SYLLABUS

Windows Programming

Objectives: To Impart the skills needed to develop windows applications using Visual C. Student will learn how to design windows and various components of windows, keyboard events, graphics and text, file handling. Student will also learn memory management techniques.

Sr. No.	Description
1.	**Windows Programming Basics:** The Advantages of Windows, How Windows Programs Work, Running Several Programs Simultaneously, Messages, An Analogy, Structure of a Windows Program, Code and Resources, Program Instances, Compiling Windows Program, Windows Memory Management.
2.	**Windows Programming:** The Windows.H, The WinMain() Function and Its Parameters, Creating the Programs Window, Messages and Adding a Message Loop, Creating a New Window Class, Message Processing Function WndProc(), Adding Custom Resource Data, Compiling the Resource Data.
3.	**Windows Controls:** Window, Types of Controls, The CreateWindow() function, Static Controls, Sending Message to a Control, C language Casts, Button Controls, Processing Button Control Messages, Button Notification Codes, List Boxes, Combo Boxes, Scroll Bars, Edit Controls.
4.	**Memory Management:** Local vs Global Memory, Local Memory Blocks, Using Fixed Memory Blocks, Changing the size of a Memory Block, Using LocalReAlloc(), Discardable Memory Blocks, Global Memory Allocation, What windows is actually doing with Memory, System Memory and System Resources.
5.	**Character Sets, Fonts, and the Keyboard:** The ANSI Character Set, Trying the Character Functions, Keyboard Message Processing, The WM_CHAR Message, System Key Messages and Dead Characters, Implementing a Simple Keyboard Interface, Selecting a Stock Font, Using Logical Fonts, Text Metric, Putting Fonts to Work, Keyboard Accelerators.
6.	**File I/O:** How Windows Programs Access Disk Files (Opening, Reading, Writing and Closing), Creating a File Selection Dialogue Box, Creating a Text Editor.
7.	**Child and Pop Up Windows:** Creating a Child Window, Sending Messages to Child Window, Fixed Child Windows, PopUp Windows.
8.	**Menus:** Creating Menus, Menus Defined as Resource Data, Creating a Menu Using the Borland Resource Workshop, Complex Menu, Creating a Menu as the Program Operates, Creating Menu Containing Bitmaps, The System Menu.
9.	**Dialog Boxes:** What is a dialogue box, How a Dialogue Box Work, Designing a Dialogue Box, Using a Dialogue Box, Exchanging Data with a Dialogue Box-Global Variable Method, Problems with using Global Variables, Exchanging Data with a Dialogue Box-Pointer Method, Modal, Modeless and System Modal Dialogue Boxes, Creating Modeless Dialogue Box.
10.	**Text & Graphics Output:** Character Mode vs Graphics Mode, The Device Context, Windows GDI, Text output, The WM_PAINT Message, Changing the Device Context, Device Context Settings, Graphics Output, Animated Graphics, The Peek Message() Loop.

CONTENT

Unit 1:	Windows Programming Basics *Kumar Vishal, Lovely Professional University*	1
Unit 2:	Windows Memory Management *Kumar Vishal, Lovely Professional University*	22
Unit 3:	Windows Programming *Kumar Vishal, Lovely Professional University*	31
Unit 4:	Windows Controls *Kumar Vishal, Lovely Professional University*	43
Unit 5:	Memory Management (I) *Anil Sharma, Lovely Professional University*	79
Unit 6:	Memory Management (II) *Anil Sharma, Lovely Professional University*	92
Unit 7:	Character Sets, Fonts and the Keyboard *Kumar Vishal, Lovely Professional University*	102
Unit 8;	File I/O *Kumar Vishal, Lovely Professional University*	125
Unit 9:	Child and Pop Up Windows *Kumar Vishal, Lovely Professional University*	140
Unit 10:	Menus *Kumar Vishal, Lovely Professional University*	160
Unit 11:	Dialog Boxes (I) *Anil Sharma, Lovely Professional University*	181
Unit 12:	Dialog Boxes (II) *Anil Sharma, Lovely Professional University*	192
Unit 13:	Windows GDI *Anil Sharma, Lovely Professional University*	208
Unit 14:	Text and Graphics Output *Anil Sharma, Lovely Professional University*	217

Notes

Unit 1: Windows Programming Basics

CONTENTS

Objectives
Introduction
1.1 The Advantages of Windows
1.1.1 Space Savings
1.1.2 Greater Accessibility and Protection for Your Data
1.1.3 Simplified Administration
1.1.4 Remote Management and Problem Analysis
1.1.5 xSeries Server Directly-attached with an Integrated xSeries Adapter (IXA)
1.1.6 Multiple Servers
1.2 How Windows Program Work
1.3 Running Several Programs Simultaneously
1.4 Messages
1.5 An Analogy
1.6 The Structure of a Windows Program
1.7 Code and Resources
1.7.1 Accessing Resources from Code
1.7.2 Creating Resources with Code
1.7.3 Different Data Types used in Resource File
1.8 Program Instances
1.9 Compiling Windows Program
1.10 Summary
1.11 Keywords
1.12 Review Questions
1.13 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concepts of windows programming basics
- Discuss advantages of windows
- Recognize how window programs work
- Discuss running several programs simultaneously
- Understand the structure of a windows program
- Explain code and resources
- Discuss program instances and compiling windows program

Notes

Introduction

Microsoft Windows is a multi-tasking operating system that permits numerous applications, pointed to here on out as processes. Every process in Windows is specified some amount of time, known as a time slice, where the application is specified the right to control the system without being interrupted by the other processes. The runtime precedence and the amount of time assigned to a process are identified by the scheduler. The scheduler is considered as the manager of this multi-tasking operating system, making sure that each process is specified the time and the priority it requires relying on the existing state of the system. Windows is what is known as an event-driven operating system. This indicates that each process is implemented depending on the events they obtain from the operating system. For instance, an application may sit at inactive and wait for the user to press a key. When that key is pressed Windows will record an event to the application that the key is down.

1.1 The Advantages of Windows

Windows environment provides most of the abilities of executing Microsoft Windows on a PC-based server and offers the following advantages over other computer systems.

1.1.1 Space Savings

There are some portions of hardware to handle requiring less physical space.

1.1.2 Greater Accessibility and Protection for Your Data

- An integrated Windows server utilizes disk storage, which is usually more dependable than PC server hard disks.
- You have access to quicker tape drives for incorporated server backups.
- Integrated servers absolutely take advantage of advanced data protection schemes which occurs in OS/400 like RAID or drive mirroring.
- You can add additional storage to incorporated servers without varying the server off.
- It is probable to achieve access to DB2® UDB for data during an improved Open Database Connectivity (ODBC) device driver by means of Access. This device driver facilitates server-to-server applications among integrated servers and OS/400.
- You have the aptitude to use an incorporated server as a second tier in a three-tier client/server application.
- Virtual networking does not entail LAN hardware and offers communications among iSeries logical partitions, Integrated Servers , and Integrated Adapters.

1.1.3 Simplified Administration

- User parameters, like passwords, are simpler to manage from OS/400. You can generate users and groups and register them from OS/400 to incorporated servers. This makes updating passwords and other user information from OS/400 simple.
- Your computer system is less complex, thanks to the incorporation of user administration function, security, server management, and backup and recovery plans among the OS/400 and Microsoft Windows environments. You can save your incorporated server data on the

Notes

similar media as other OS/400 data and reinstate individual files in addition to OS/400 objects.

1.1.4 Remote Management and Problem Analysis

- You can sign-on to OS/400 from a distant location and shut down or restart your incorporated server.
- As you can mirror incorporated server event log information to OS/400 you can distantly examine Microsoft Windows errors.

1.1.5 xSeries Server directly-attached with an Integrated xSeries Adapter (IXA)

- You have significantly more suppleness in configuring a full size xSeries than you have in configuring an IXS, an xSeries on a card. The full size xSeries can then be directly linked to the iSeries with an IXA.
- Full size xSeries models are released more frequently, meaning that you can obtain the most up-to-date processors and other hardware.
- More PCI trait cards are obtainable for full size xSeries than for IXSs.

1.1.6 Multiple Servers

- Cluster service permits you to attach multiple servers into server clusters. Server clusters give high-availability and simple manageability of data and programs running inside the cluster.
- By not using LAN hardware, servers and logical partitions running on the similar iSeries have high-performance, protected virtual networking communications.
- You can execute numerous integrated servers on a single iSeries. Not only suitable and efficient, this also provides you the aptitude to easily switch to another up-and-running server if the hardware fails.
- If you have numerous integrated servers installed on your iSeries, you can describe their Windows domain roles in a manner that will abridge user enrollment and access.

Example: You would like to set up one of these servers as a domain controller. Then you only have to register users to the domain controller and users can log on from any Microsoft Windows machine on that domain.

- An iSeries's optical and tape drives can be shared with incorporated servers executing on the iSeries.

Self Assessment

Fill in the blanks:

1. ..... is a multi-tasking operating system that permits numerous applications, pointed to here on out as processes.
2. An integrated Windows server utilizes disk storage, which is usually more dependable than PC server ..... .

Notes

1.2 How Windows Program Work

To generate a basic application, you will initially require a compiler that executes on a Microsoft Windows operating system. Even though you can apply Win32 on numerous languages involving Pascal (namely Borland Delphi), we will utilize only one language. Actually the Win32 library is written in C, which is also the main language of the Microsoft Windows operating systems.

Generating a Win32 Program

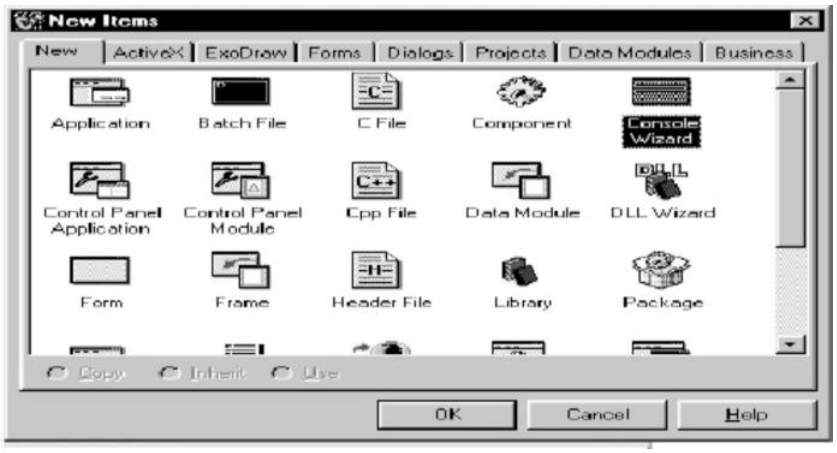
All Win32 programs chiefly appear the same and act the same but, just like C++ programs, there are small differences in terms of forming a program, relying on the compiler you are utilizing. Here we will be testing our programs on Borland C++ Builder, Microsoft Visual C++, and Microsoft Visual C++.NET.

For a fundamental Win32 program, the contents of a Win32 program are similar. You will feel a dissimilarity only when you begin adding some objects known as resources. To create a Win32 program by means of Borland C++ Builder, you must generate a console application by means of the Console Wizard. You must confirm you don't choose any alternative from the Console Wizard dialog box. After clicking OK, you are presented with a semi-empty file that contains only the inclusion of the windows.h library and the WinMain() function declaration. From there, you are prepared. From most surroundings used, Borland C++ builder is the only one that offers the simplest, but also unluckily the emptiest template to generate a Win32 application. It doesn't offer any real template nor any aid on what to do with the specified file. In protection of the Borland C++ Builder, as you will observe with Microsoft Visual C++ and the other environments may fill your file with statements you don't require, you don't like, or you don't want.

Notes Borland C++ Builder offers the empty file so you can liberally decide how you want to generate you program and what you want to comprise in your program. This signifies that we agree with Borland C++ Builder offering an empty file since at least the syntax of the WinMain() function is provided to you.

Practical Learning: Windows Programming

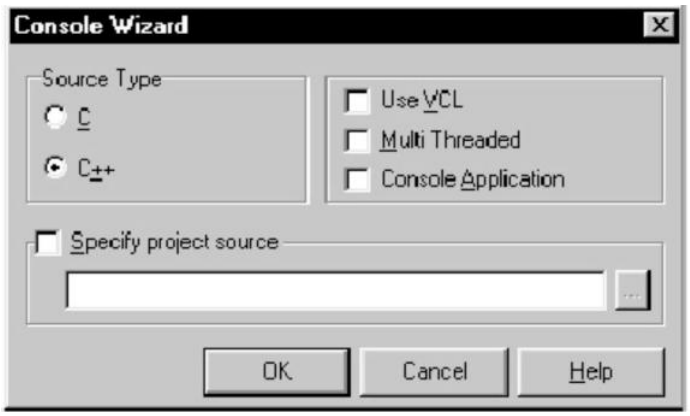
1. Begin Borland C++ Builder
2. On the main menu, click File-> New... or File -> New -> Other...



3. In the New Items dialog box, click Console Wizard and click OK

Notes

4. In the Console Wizard, confirm that only the C++ radio button is chosen:



5. Click OK. You are offered with a file as below:

```
//---------------------------------------
#include <windows.h> #pragma hdrstop
//---------------------------------------
#pragma args n used
WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{return 0;
}
//-----------------------------------------
```

6. Save the application in a new folder named Win32A.
7. Save the first file as Main.cpp and save the project as SimpleWindow.

Using Microsoft Visual C++

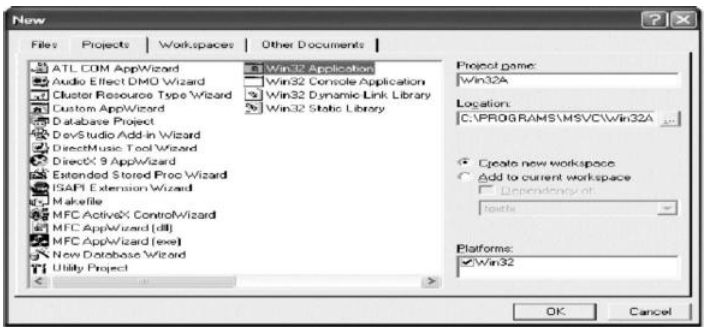
To create a Win32 application by means of Microsoft Visual C++, exhibit the New (5 and 6 versions) or New Project (.Net version) dialog box and choose Win32 application (5 and 6) or Win32 Project (.Net) item. Microsoft Visual C++ offers the fastest and quite most complete means of creating a Win32 application.

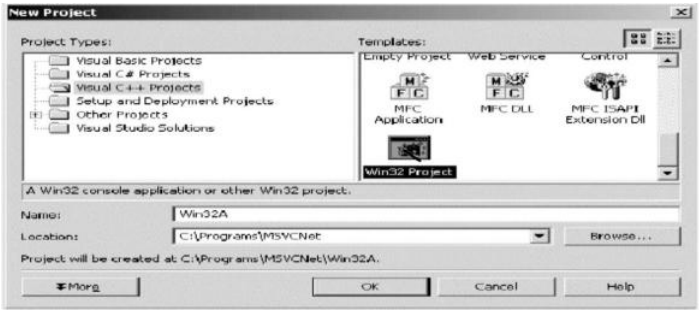
Example: It offers a skeleton application with all of the code a fundamental application would need. Since we are learning Win32, we will go the hard manner, which includes creating an application from scratch.

Actually, this permits to give approximately (but not exactly) the similar instructions as Borland C++ Builder.

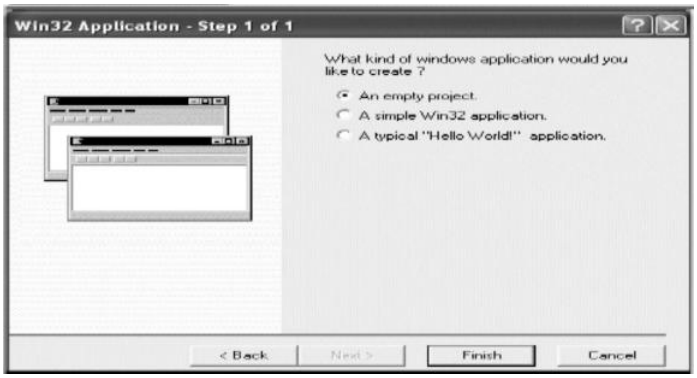
1. Start the Microsoft Development Environment.
2. On the main menu, click File -> New... or File -> New -> Project...
3. In the New or New Project dialog box, click either Win32 Application or click Visual C++ Projects and Win32Project:

Notes

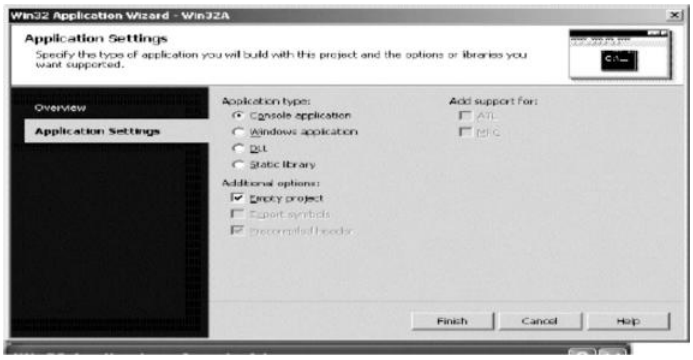




4. In the location, type the path where the application should be amassed, like C:\Programs\MSVC.
5. In the Name edit box, type the name of the application as Win32A and click OK.
6. In the next dialog box of the wizard, if you are using MSVC 5 or 6, click the An Empty Project radio button:



If you are using MSVC.Net, click Application Settings, then click the Console Application radio button, then click the Empty Project check box:



Notes

7. Click Finish. If you are using MSVC 6, you will be offered with another dialog box; in this case click OK.
8. To create the first required file of the program, if you are using MSVC 5 or 6, on the main menu, click File -> New. If you are using MSVC.Net, on the main menu, click Project -> Add New Item...
9. If you are using MSVC.Net, confirm that Visual C++ is chosen in the Categories tree view. In both cases click either C++ Source File or C++ File (.cpp)

Self Assessment

Fill in the blanks:

3. To generate a basic application, you will initially require a ..... that executes on a Microsoft Windows operating system.
4. For a fundamental Win32 program, the contents of a Win32 program are ..... .

1.3 Running Several Programs Simultaneously

If you function with **numerous programs** simultaneously you know how tedious is to **run and launch** them one by one! Actually you require to find them between the other applications you have installed on your Window, click their icons and linger for them to open. This operation is quite unbearable. That's why you require a particular shortcut (batch file) which is able to start all of them in one click! This trick will let you attain a great, time saving result. Opening several applications in a matter of a couple seconds! Stop browsing your computer folders, stop looking for the right icon. Handle everything from the same place.

1. Click **Start.**
2. Click **All Programs.**
3. Click **Accessories.**
4. Click **Notepad** to open it.
5. Now write the following code:

Start "" (confirm to leave a space before and after "") followed by the absolute path of the program you desire to open in quote.

Example: Start ""

"C:\Users\YourName\AppData\Local\Google\Chrome\Application\chrome.exe"

6. Right after that press Enter and write another line like the one above so as to open a new application.
7. Ensure to write each Start "" command on a new line so that a line will enclose a Start "" command only, or else the batch file won't work and you won't be able to *open numerous programs*!
8. Now save the file with any name you desire and confirm to save it as .bat extension (and not as .txt).

Notes

Caution If you do not recognize the absolute path of an application on your computer, right click its icon positioned on the desktop – Properties – copy the path situated in the Destination field.

Task Write the steps for Running Several Programs Simultaneously.

Self Assessment

Fill in the blank:

5. If you do not recognize the absolute path of an application on your computer, right click its icon positioned on the desktop ..... copy the path situated in the Destination field.

1.4 Messages

A computer application is prepared with Windows controls that permit the user to interrelate with the computer. Each control generates messages and sends them to the operating system. To supervise these messages, they are managed by a function pointer known as a Windows procedure. This function can emerge as follows:

```
LRESULT CALLBACK MessageProcedure(HWND hWnd, UINT uMsg, WPARAM wParam,
LPARAM lParam);
```

This function utilizes a **switch** control to list all essential messages and process each one in turn. This processes just the messages that you ask it to. If you have available messages, and you will always have unrefined messages, you can call the DefWindowProc() function at the end to take over.

The most essential message you can process is to ensure a user can close a window after using it. This can be completed with a function known as **PostQuitMessage()**. Its syntax is:

```
VOID PostQuitMessage(int nExitCode)
```

This function comprises one argument which is the value of the **LPARAM** argument. To close a window, you can pass the argument as **WM_QUIT**.

Depending on this, a simple Windows procedure can be defined as below:

```
LRESULT CALLBACK WndProcedure(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM
lParam)
{
      switch(Msg)
     {
      case WM_DESTROY:
            PostQuitMessage(WM_QUIT);
          break;
      default:
            return DefWindowProc(hWnd, Msg, wParam, lParam);
     }
```

Notes

```
    return 0;
}
```

Task Illustrate the use of PostQuitMessage().

Self Assessment

Fill in the blanks:

6. Each window control generates ..... and sends them to the operating system.
7. If you have available messages, and you will always have unrefined messages, you can call the ..... function at the end to take over.

1.5 An Analogy

Windows programming is complicated. Windows class libraries make Windows programming simpler. True or false?

```
 bool IsWinProgEasier (Method method)
{
      if (method == WIN_CLASS_LIBRARIES)
         return false;
    else
         return true;
}
```

Sincerely, if you want to write programs that are precisely like the inventors of MFC or OWL figured out you would, and you don't care regarding the overhead, then by means of class libraries and application wizards is the method to go. But any time you desire to step outside of this path, you'll find yourself in profound problem.

Let us provide you an analogy. Visualize that you're purchasing a set of Lego blocks. You can obtain a general purpose set, or you can purchase a particular set for structuring a pirate ship. If you want to do to build a pirate ship, the second option is superior. But if you attempt to make a Lego car, you'll have to conquer a few troubles. Finally you'll come up with something that looks like a car, but it will have a humorous steering wheel, utilize an anchor for breaking, and the driver will have a stick leg and a black patch over his eye.

After functioning with this Pirate Lego set for a while, you will turn out to be a specialist in constructing approximately anything that could be built with a common purpose set. You'll learn all the tricks — like how to remove the patch from the pirate's eye, how to paint over the skull and crossbones, etc. Eventually you'll arrive at a point when the amount of information you've assimilated regarding the Pirate set will be much more than the fundamental engineering principles you'd have to learn so as to use general purpose Lego. From that point on, you'll be throwing good money after bad money, investing in growingly complicated (and complicated) Pirate sets.

Notes

Self Assessment

Fill in the blanks:

8. To write programs that are precisely like the inventors of MFC or OWL figured out you would, and you don't care regarding the overhead, then using class libraries and ..... is the method to go.

9. A ..... function is called by Windows to process messages for the application.

1.6 The Structure of a Windows Program

For a negligible Windows program that just utilizes the Windows API, you will write two functions. These are a WinMain() function, where execution of the program starts and basic program initialization is performed, and a WindowProc() function that is called by Windows to process messages for the application. The WindowProc() part of a Windows program is typically the larger portion since this is where most of the application-specific code is, reacting to messages caused by user input of one type or another.

Notes Even though these two functions make up an absolute program, they are not directly associated. WinMain() does not call WindowProc(), Windows does. Actually, Windows also calls WinMain().

main.c

```
#include <windows.h>


//The Message Processing Function

LRESULT WINAPI WindowProc(HWND hWnd, UINT message,WPARAM wParam, LPARAM
lParam);


//The entry point for a Windows application

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, /*Always
NULL*/

                      LPSTR lpCmdLine, int nCmdShow)

{

   static LPCTSTR szAppName = L"MyApp";      // Define window class name


   WNDCLASSEX WindowClass;       // Structure to hold our window's attributes


   WindowClass.cbSize = sizeof(WNDCLASSEX);           // Set structure size

   WindowClass.style    = CS_HREDRAW | CS_VREDRAW;    // Redraw the window if
the size changes

   WindowClass.lpfnWndProc = WindowProc;     // Define the message handling
function

  WindowClass.cbClsExtra = 0;         // No extra bytes after the window class

   WindowClass.cbWndExtra = 0;          // structure or the window instance
```

Notes

```
   WindowClass.hInstance = hInstance;
    // Application instance handle
   WindowClass.hIcon = LoadIcon(0, IDI_APPLICATION);
    // Set default application icon
   WindowClass.hCursor = LoadCursor(0, IDC_ARROW);
   // Set window cursor to be the standard arrow
   WindowClass.hbrBackground =
         static_cast<HBRUSH>(GetStockObject(GRAY_BRUSH));
   // Set gray brush for background color
   WindowClass.lpszMenuName = 0;          // No menu
   WindowClass.lpszClassName = szAppName;        // Set class name
   WindowClass.hIconSm = 0;                       // Default small icon

   // Now register our window class
   RegisterClassEx(&WindowClass);

   HWND hWnd;          // Window handle
   // Now we can create the window
   hWnd = CreateWindow(
            szAppName,         // the window class name
            L"A Basic Window the Hard Way",   // The window title
            WS_OVERLAPPEDWINDOW,       // Window style as overlapped
            CW_USEDEFAULT,         // Default screen position of  upper left
            CW_USEDEFAULT,         // corner of our window as x,y
            CW_USEDEFAULT,         // Default window size.
            CW_USEDEFAULT,           // CW_USEDEFAULT only applies to windows
                                       specified as WS_OVERLAPPED
           0,                        // No parent window
           0,                        // No menu
            hInstance,          // Program Instance handle
           0                          // No window creation data
         );

   ShowWindow(hWnd, nCmdShow);       // Display the window
   UpdateWindow(hWnd);                // Cause window client area to be
drawn

   MSG msg;               // Windows message structure
   // The message loop.
   //Rretrieves messages that are queued for the application.
while(GetMessage(&msg, 0, 0, 0) == TRUE)   // Get any messages.
        //WM_QUIT causes it to return false.
        //When error occurs the return value is -1.
```

Notes

```
    {
        TranslateMessage(&msg);
        //   Translates virtual-key messages into character messages.
        DispatchMessage(&msg); // Dispatch the message to Windows
                                  and let Windows call the WndProc() function.
    }

  return static_cast<int>(msg.wParam);// End, so return to Windows
}
```

//The Message Processing Function

Called by Windows to manage all the messages for a specified window that aren't queued, which involves those initiated in the message loop in WinMain().

Did u know? The queued messages are mostly those caused by user input from either the mouse or the keyboard.

The non-queued messages for which Windows calls your WindowProc() function directly, are either messages that your program formed, usually as a result of getting a message from the queue and then dispatching it, or messages that are related with window management—like managing menus and scrollbars, or resizing the window.//

```
LRESULT WINAPI WindowProc(HWND hWnd, UINT message,
                          WPARAM wParam, LPARAM lParam)
{
  HDC hDC;                              // Display context handle
   PAINTSTRUCT PaintSt;    // Structure defining area to be drawn
  RECT aRect;                      // A working rectangle
  switch(message)                  // Process selected messages
  {
    case WM_PAINT:                     // Message is to redraw the window

      hDC = BeginPaint(hWnd, &PaintSt);     // Prepare to draw the window
      GetClientRect(hWnd, &aRect);     // Get upper left and lower right
                                          of client area
      SetBkMode(hDC, TRANSPARENT);    // Set text background mode

      // Now draw the text in the window client area
      DrawText(
             hDC,                          // Device context handle
              L"Hello, Windows!",
             -1,                         // Indicate null terminated string
              &aRect,             // Rectangle in which text is to be drawn
            DT_SINGLELINE |     DT_CENTER | DT_VCENTER);     // Text format
```

Notes

```
        EndPaint(hWnd, &PaintSt);            // Terminate window redraw
                                                operation
      return 0;

    case WM_DESTROY:                     // Window is being destroyed
        PostQuitMessage(0);
      return 0;

    default:                          // Any other message - call default message
                                        processing
        return DefWindowProc(hWnd, message, wParam, lParam);
  }
}
```

Self Assessment

Fill in the blanks:

10. In ..... function, execution of the program starts and basic program initialization is performed.
11. A ..... function is called by Windows to process messages for the application.

1.7 Code and Resources

Resources are defined as the data that you can include to the applications executable file resources can be:

- ***standard:*** icon, cursor, menu, dialog box, bitmap, enhanced metafile, font, accelerator table, message-table entry, string-table entry, or version.
- ***custom:*** any sort of data that doesn't fall into the preceding category (for instance a mp3 file or a dictionary database).

Add two new files to your project. Name them resource.h and resource.rc.

Caution Resource.rc will enclose the resource definitions and resource.h will define constants.

This overview focuses on how Windows Presentation Foundation (WPF) resources can be accessed or created by means of code instead of Extensible Application Markup Language (XAML) syntax.

1.7.1 Accessing Resources from Code

The keys that recognize resources if they are defined during XAML are also used to recover particular resources if you demand the resource in code. The simplest manner to recover a resource from code is to call either the FindResource or the TryFindResource method from framework-level objects in your application. The behavioral dissimilarity between these methods is what occurs if the needed key is not found. FindResource elevates an exception; TryFindResource will not raise an exception but returns null.

Notes

Did u know? Each method takes the resource key as an input parameter, and returns a loosely typed object.

Typically, a resource key is a string, but there are infrequent non-string usages; see the Using Objects as Keys section for details. Usually you would cast the returned object to the type required by the property that you are setting when requesting the resource.

Caution The lookup logic for code resource resolution is the similar as the dynamic resource reference XAML case.

The hunt for resources starts from the calling element, then persists to successive parent elements in the logical tree. The lookup continues onwards into application resources, themes, and system resources if essential. A code demand for a resource will properly account for runtime modifies in resource dictionaries that might have been made succeeding to that resource dictionary being loaded from XAML, and also for real-time system resource m.

Example: The following is a concise code example that locates a resource by key and utilizes the returned value to put a property, executed as a Click event handler.

```
C#

void SetBGByResource(object sender, RoutedEventArgs e)
{
   Button b = sender as Button;
   b.Background = (Brush)this.FindResource("RainbowBrush");
}
```

1.7.2 Creating Resources with Code

If you want to generate a whole WPF application in code, you might also desire to create any resources in that application in code. To attain this, create a new Resource Dictionary instance, and then add all the resources to the dictionary by means of successive calls to Resource Dictionary.Add. Then, use the Resource Dictionary thus generated to set the Resources property on an element that is present in a page scope, or the Application.Resources. You could also preserve the Resource Dictionary as a separate object without adding it to an element. On the other hand, if you do this, you must access the resources within it by item key, as if it were a generic dictionary. A Resource Dictionary that is not associated to an element Resources property would not occur as part of the element tree and has no scope in a lookup sequence that can be used by Find Resource and associated methods.

1.7.3 Different Data Types used in Resource File

Microsoft Windows applications often depend on files that contain non-executable data, such as Extensible Application Markup Language (XAML), images, video, and audio. Windows Presentation Foundation (WPF) offers special support for configuring, identifying, and using these types of data files, which are called application data files. This support revolves around a specific set of application data file types, including:

Notes

- ***Resource Files:*** Data files that are compiled into either an executable or library WPF assembly.
- ***Content Files:*** Standalone data files that have an explicit association with an executable WPF assembly.
- ***Site of Origin Files:*** Standalone data files that have no association with an executable WPF assembly.

One important distinction to make between these three types of files is that resource files and content files are known at build time; an assembly has explicit knowledge of them. For site of origin files, however, an assembly may have no knowledge of them at all, or implicit knowledge through a pack uniform resource identifier (URI) reference; the case of the latter, there is no guarantee that the referenced site of origin file actually exists.

Resource Files

If an application data file must always be available to an application, the only way to guarantee availability is to compile it into an application's main executable assembly or one of its referenced assemblies. This type of application data file is known as a resource file.

You should use resource files when:

- You don't need to update the resource file's content after it is compiled into an assembly.
- You want to simplify application distribution complexity by reducing the number of file dependencies.
- Your application data file needs to be localizable (see WPF Globalization and Localization Overview).

Configuring Resource Files

In WPF, a resource file is a file that is included in a Microsoft build engine (MSBuild) project as a Resource item.

```
<Project "xmlns=http://schemas.microsoft.com/developer/msbuild/2003" ...
>
  ...
  <ItemGroup>
     <Resource Include="ResourceFile.xaml" />
  </ItemGroup>
  ...
</Project>
```

Notes In Microsoft Visual Studio, you create a resource file by adding a file to a project and setting its Build Action to Resource.

When the project is built, MSBuild compiles the resource into the assembly.

Notes

Using Resource Files

To load a resource file, you can call the GetResourceStream method of the Application class, passing a pack URI that identifies the desired resource file.GetResourceStream returns a StreamResourceInfo object, which exposes the resource file as a Stream and describes its content type.

As an example, the following code shows how to use GetResourceStream to load a Page resource file and set it as the content of a Frame (pageFrame):

```
C#

VB

// Navigate to xaml page

Uri uri = new Uri("/PageResourceFile.xaml", UriKind.Relative);

StreamResourceInfo info = Application.GetResourceStream(uri);

System.Windows.Markup.XamlReader reader = new
System.Windows.Markup.XamlReader();

Page page = (Page)reader.LoadAsync(info.Stream);

this.pageFrame.Content = page;
```

While calling GetResourceStream gives you access to the Stream, you need to perform the additional work of converting it to the type of the property that you'll be setting it with. Instead, you can let WPF take care of opening and converting the Stream by loading a resource file directly into the property of a type using code.

Self Assessment

Fill in the blanks:

12. ..... are defined as the data that you can include to the applications executable file.
13. The simplest manner to recover a resource from code is to call either the ..... or the TryFindResource method from framework-level objects in your application.

1.8 Program Instances

A window is considered as parent when it can be used to host, hold, or take other windows. For instance, when the computer begins, it draws its main screen, also known as the desktop, which covers the widest area that the monitor screen can propose. This primary window turns out to be the host of all other window that will exhibit as long as the computer is own. This desktop is also a complete window in its own right. As declared already, to get its handle, you can call the **GetDesktopWindow()** function.

After the desktop has been generated, a window of yours can display if the user begins your application. This signifies that an application must have been generated for the user to use it. When the user opens an application, we also say that the application has been instantiated or an instance of the application has been generated. Depending on this, any time you generate an application, you must offer an instance of it. This permits the operating system to handle your application with respect to its communication with the user and also its relationship with other resources. Thus, you must always generate an instance for your application. This is taken care of by the first argument of the **WinMain()** function.

Notes

If an application has already been formed, to obtain its instance, you can call the **GetWindowLong()** function. Its syntax is:

```
LONG GetWindowLong(HWND hWnd, int nIndex);
```

Even though this function is used for many other reasons, it can also aid you get the instance of an application. To perform this, pass the first argument as the handle to a window of the application you are probing and pass the second argument as **GWL_HINSTANCE**.

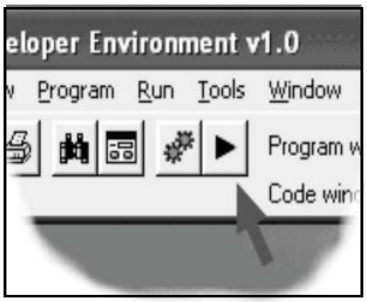
Self Assessment

Fill in the blank:

14. If an application has already been formed, to obtain its instance, you can call the ..... function.

1.9 Compiling Windows Program

Click the Run button (displayed below with the arrow) and wait a few seconds. This compiles the program to an EXE file and runs it:



When the program runs, you will observe the dialog on the screen. It appears like this:

You have effectively written a Windows program that could execute under any version of Windows. The window above is portion of the EXE file, and it can outside the Developer Environment if required. Press the "Close me" button to finish the program. The Close button triggers the Command event, which unloads the dialog. The program will then end since it has no windows left.

The resource data is stored in separate file in windows programs, compiling a DOS program is not more involved then the process of putting together a complete Windows program. Figure 1.1 shows that how a C program is run under DOS, how it is being compiled and linked to create a finished program. Under DOS, the C compiler converts the source code file into an objective file. A linker converts the objective file into the finished executable program. In this way a source code can be converted to the finished program with the help of compiler and linker.

Notes

Figure 1.1: Steps in Creating a DOS Program

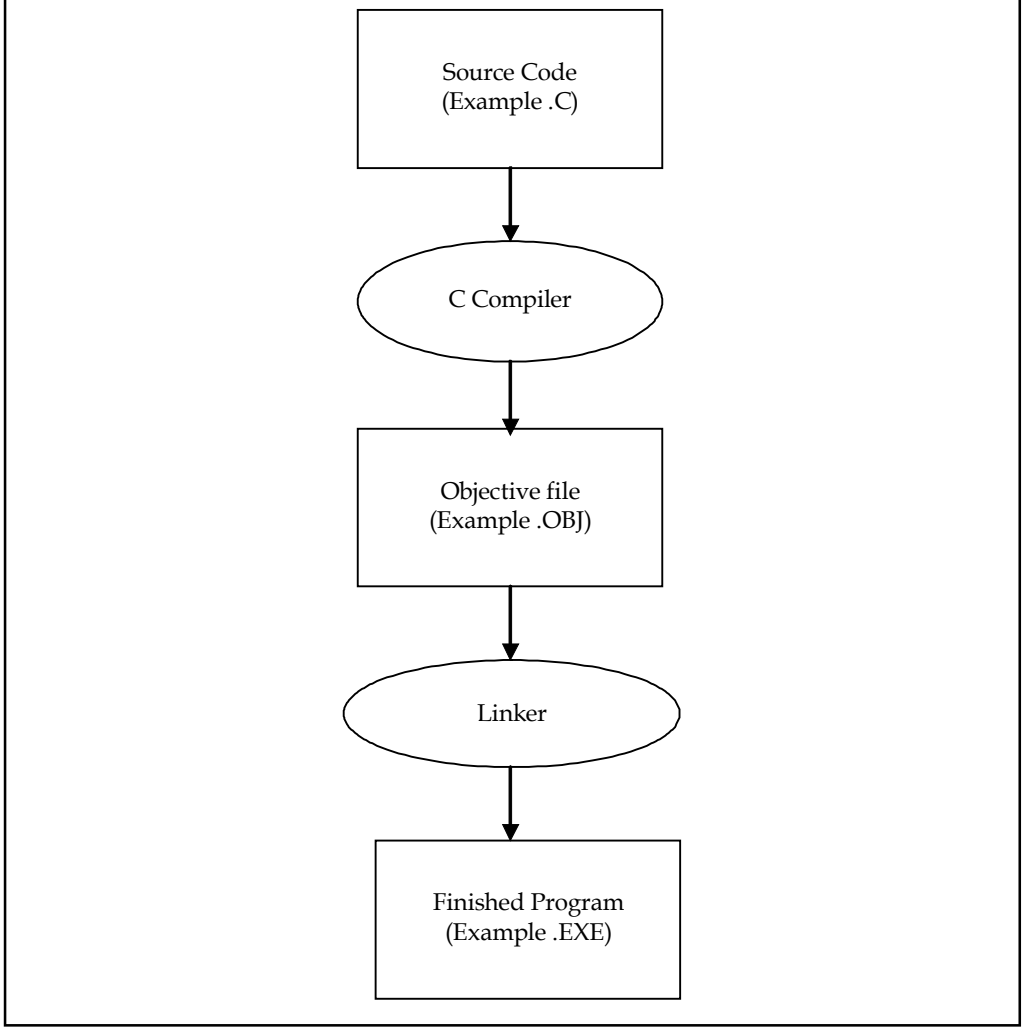
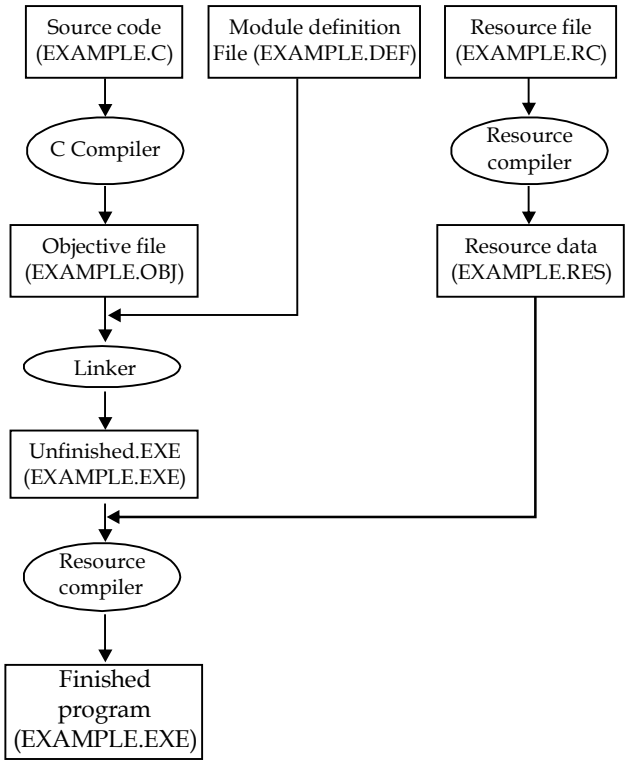


Figure 1.2 shows a flow diagram for the creation of a Windows programs. In this figure the source code file gets converts to objective file from the compilers same as in the DOS. In windows programs the linker gets a few additional information from a small file called the "module definition file" with the file name extension ".DEF". This file tells the linker how to assemble the program. The linker combines the module definition file information and the object file to make an unfinished .EXE file. The unfinished .EXE file lacks the resource data.

The main difference between Windows programs and DOS programs is in the compilation of the resource data file with the extension of ".RES". In DOS programs there is no resource data but in windows program the resource data is added to the unfinished.EXE file to create the finished executable program. The resource data is basically stuck onto the end of the program's code and becomes part of the programs file. In addition to adding the resource data the resource compiler writes the Windows version number into the program file.

Notes

Figure 1.2: Steps in Creating a Windows Program



Self Assessment

Fill in the blank:

15. The ..... button triggers the Command event, which unloads the dialog.

1.10 Summary

- Microsoft Windows is a multi-tasking operating system that permits numerous applications, pointed to here on out as processes.
- An integrated Windows server utilizes disk storage, which is usually more dependable than PC server hard disks.
- All Win32 programs chiefly appear the same and act the same but, just like C++ programs, there are small differences in terms of forming a program, relying on the compiler you are utilizing.
- A computer application is prepared with Windows controls that permit the user to interrelate with the computer. Each control generates messages and sends them to the operating system.
- To supervise the messages, they are managed by a function pointer known as a Windows procedure.

Notes

- In WinMain() function, execution of the program starts and basic program initialization is performed.
- A WindowProc() function is called by Windows to process messages for the application.
- Resources are defined as the data that you can include to the applications executable file.

1.11 Keywords

Resources: Resources are defined as the data that you can include to the applications executable file.

WindowProc(): A WindowProc() function is called by Windows to process messages for the application.

Windows: Microsoft Windows is a multi-tasking operating system that permits numerous applications, pointed to here on out as processes.

WinMain() Function: In WinMain() function, execution of the program starts and basic program initialization is performed.

1.12 Review Questions

1. What are Microsoft windows? Illustrate the advantages of windows.
2. Explain how windows program works.
3. Elucidate the working of Windows Program by using visual c++.
4. Illustrate how you will handle running several programs simultaneously.
5. What are windows messages? Illustrate.
6. Write the syntax of PostQuitMessage(). Also discuss parameters.
7. Discuss the Structure of a Windows Program with example.
8. Illustrate the process of accessing Resources from Code
9. After the desktop has been generated, a window of yours can display if the user begins your application. Comment.
10. Illustrate the steps for Compiling Windows Program.

Answers: Self Assessment

1. Microsoft Windows
2. Hard Disks
3. Compiler
4. Similar
5. Properties
6. Messages
7. DefWindowProc()
8. Application Wizards
9. WindowProc()
10. WinMain()
11. WindowProc()
12. Resources
13. FindResource
14. GetWindowLong()
15. Close

1.13 Further Readings

Notes

Books

Brent E. Rector, *Win32 Programming*, Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

www.apitalk.com/windows-Programming

Notes

Unit 2: Windows Memory Management

CONTENTS

Objectives

Introduction

2.1 Windows Memory Management

2.2 Paging in x86 Processor

2.2.1 Windows Page Table Management

2.3 Memory Protection

2.4 Windows Logical Memory Layout

2.5 Summary

2.6 Keywords

2.7 Review Questions

2.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of Windows Memory Management
- Discuss Paging in x86 Processor
- Illustrate memory protection

Introduction

Memory management in Windows operating systems has developed into an affluent and classy architecture, competent of scaling from the minute embedded platforms (where Windows implements from ROM) all the way up to the multi-terabyte NUMA configurations, taking full benefit of all capabilities of obtainable and upcoming hardware designs.

With every release of Windows, memory management assists many new traits and capabilities. Advances in algorithms and techniques capitulate a affluent and classy code base, which is sustained as a single code base for all platforms and SKUs.

2.1 Windows Memory Management

Windows on 32 bit x86 systems can utilize up to 4GB of physical memory. This is because of the fact that the processor's address bus which is 32 lines or 32 bits can only utilize address range from 0x00000000 to 0xFFFFFFFF which is 4GB. Windows also permits every process to have its own 4GB logical address space. The lower 2GB of this address space is available for the user mode process and upper 2GB is set aside for Windows Kernel mode code. How does Windows provide 4GB address space each to numerous processes when the total memory it can utilize is also restricted to 4GB. To attain this Windows uses a feature of x86 processor (386 and above) called paging.

Notes

Did u know? Paging permits the software to use a diverse memory address (called logical address) than the physical memory address.

The Processor's paging unit converts this logical address to the physical address transparently. This permits every process in the system to have its own 4GB logical address space. To recognize this in more details, let us first have a look at how the paging in x86 functions.

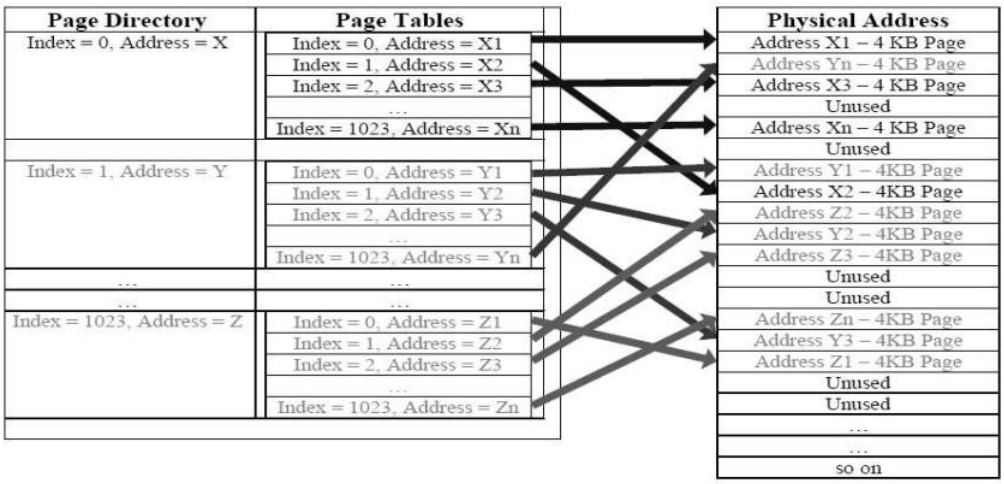
Self Assessment

Fill in the blanks:

1. Windows on 32 bit x86 systems can utilize up to ..... of physical memory.
2. Windows permits every process to have its own 4GB ..... address space.
3. Paging permits the software to use a diverse memory address (called logical address) than the ..... memory address.
4. The Processor's ..... unit converts this logical address to the physical address transparently.

2.2 Paging in x86 Processor

The x86 processor separates the physical address space (or physical memory) in 4 KB pages. Therefore to address 4GB of memory, we will require 1 Mega (1024x1024) 4KB pages. The processor utilizes a two level structure to refer to these 1 Mega pages. You can consider of it as a two dimensional matrix of 1024x1024 elements. The first dimension is called Page Directory and second dimension is called Page Table. So we can generate 1 Page directory with 1024 entries, each of which refers to a Page Table. This will permit us to have 1024 page tables. Each page table in turn can have 1024 entries, each of which points to a 4 KB page. Graphically it appears something like:



Every Page Directory Entry (or PDE) is 4 bytes in size and refers to a Page Table. Likewise each Page Table Entry (or PTE) is 4 bytes and points to a physical address of a 4KB page. To store 1024 PDE each containing 1024 PTE, we will need a total memory of 4 x 1024 x 1024 bytes i.e. 4MB. Thus to divide the whole 4GB address space into 4 KB pages, we need 4 MB of memory.

Notes

As discussed above, the whole address space is divided in 4KB pages. So when a PDE or PTE is used, its upper 20 bits gives a 4KB page aligned address and lower 12 bits are utilized to store the page defense information and some other housekeeping information needed by an OS for proper functioning.

Notes The upper 20 bits which shows the actual physical address are called Page Frame Number (or PFN). Details on defense bits and other bits in the lower 12 bits can be located in here.

Task Make distinction between page directory and page table.

2.2.1 Windows Page Table Management

In windows every process has its own Page Directory and Page Tables. Therefore windows assign 4 MB of this space per process. When a process is formed, every entry in Page Directory includes physical address of a Page Table. The entries in the page table are either valid or invalid. Valid entries enclose physical address of 4KB page assigned to the process. An invalid entry includes some particular bits to mark it invalid and these entries are called Invalid PTEs. As the memory assigned by the process, these entries in Page Table are filled with the physical address of the assigned pages. You should keep in mind one thing here that a process doesn't recognize anything regarding physical memory and it only utilizes logical addresses. The details of which logical address corresponds to which physical address is handled evidently by Windows Memory manager and the processor. The address at which the page directory of a process is positioned in physical memory is pointed to as Page Directory Base address. This Page Directory Base address is amassed in a special CPU register called CR3 (on x86). On context switch, Windows loads the new value of CR3 to point to the new process's Page Directory Base. This manner every process gets its own division of the whole 4GB physical address space. Certainly the total memory assigned at one time to all the process's in a system cannot go beyond the total amount of RAM + page file size but the method discussed above permits windows to give every process its own 4GB logical (or Virtual) address space. We call it Virtual Address space since although the process has the whole 4GB address range obtainable to it, it can only use the memory which is assigned to it. Compilers can generate a program that depends on the code being at an *accurate* location in memory, every time it is executed.

Example: With virtual memory, the process *considers,* it is at 0x080482a0, but in fact it could be at physical memory position 0x1000000.

If a process attempts to access an unallocated address, it will obtain an access violation since the PTE subsequent to that address refers to an invalid value. Also the process can't assign more memory than what is obtainable in the system. This way of separating logical memory from physical memory has many benefits. A process gets a linear 4GB address space so application programmers don't have to concern regarding segments and all unlike in old DOS days. It also permits windows to run multiple processes and let them employ physical memory on a machine without worrying about them stomping on each other's address space. A logical address in one process will never refer to the physical memory assigned to another process (unless they are utilizing some sort of shared memory). Therefore, one process can never read from or write to another process's memory.

Notes

The translation from logical to physical address is performed by the processor. A 32 bit logical address is separated into three parts as displayed below. The processor loads the physical address of the page directory base amassed in CR3. It then utilizes the upper 10 bits from the logical address as an index in the Page directory. This provides the processor a page directory entry (PDE) which refers to a Page Table. The next 10 bits are used as an index in the page table. By means of these 10 bits, it obtains a page table entry (or PTE) which points to a 4KB physical page.

10 bits	10 bits	12 bits

Did u know? The lowest 12 bits are utilized to address the individual bytes on a page.

Self Assessment

Fill in the blanks:

5. The ..... separates the physical address space (or physical memory) in 4 KB pages.
6. The first dimension of X86 processor is called ..... and second dimension is called Page Table.
7. We can generate 1 Page directory with 1024 entries, each of which refers to a ..... .
8. Every ..... is 4 bytes in size and refers to a Page Table.
9. When a process is formed, every entry in Page Directory includes ..... of a Page Table.
10. ..... entries enclose physical address of 4KB page assigned to the process.
11. An ..... entry includes some particular bits to mark it invalid and these entries are called Invalid PTEs.

2.3 Memory Protection

Windows offers memory protection by means of the virtual memory hardware. The accomplishment of this protection differs with the processor.

Example: Code pages in the address space of a process can be marked read-only and protected from alteration by user-mode threads.

Windows give memory protection to all the processes in order that one process can't utilize other process's memory. This makes sure smooth operation of multiple processes concurrently. Windows make sure this protection by doing following:

- It only puts the physical address of assigned memory in PTE for a process. This makes sure that the process's obtains an access violation if it attempts to access an address which is not allocated.
- A rouge process may attempt to adjust its page tables so that it can access the physical memory belonging to another process. Windows defend this sort of attacks by accumulating page tables in kernel address space. Remember from our previous discussion that out of the 4GB logical address space specified to a process, 2GB is given to user mode and 2GB is reserved for windows kernel.

Notes

So a user mode application can not directly access or alter the page tables. Certainly if a kernel mode driver wants to do that, it can do it since once you are in kernel mode, you almost own the whole system. To recognize this in more details, study the next section on Windows Logical memory layout.

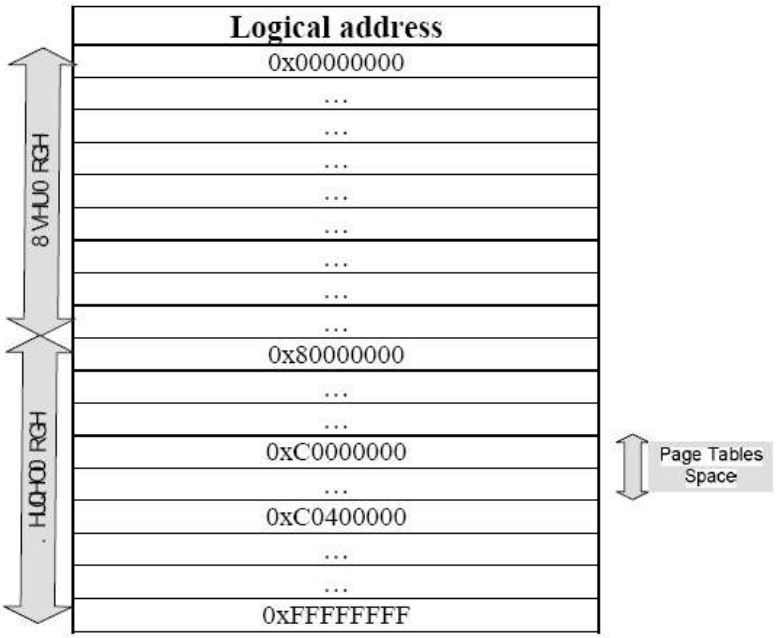
Self Assessment

Fill in the blanks:

12. Windows give memory ..... to all the processes in order that one process can't utilize other process's memory.
13. ..... application can not directly access or alter the page tables.

2.4 Windows Logical Memory Layout

Windows provides lower 2GB (or 3GB relying upon boot.ini switch) logical address space of a process to user mode and upper 2GB (or 1GB relying upon boot.ini switch) to Windows kernel. Out of the total kernel address space, it reserves addresses from 0xC0000000 to 0xC03FFFFF for Page Tables and Page Directory. Each process has its Page Tables positioned at the **logical address 0xC0000000** and page directory situated at **logical address** 0xC0300000. This logical memory arrangement is displayed below:



You can utilize Windows kernel debugger kd or windbg to verify this (point to !pte and !vtop debugger extensions). The physical address to this page directory is amassed in CR3. The 1024 addresses beginning from 0xC0300000 displays Page Directory Entry (PDE). Each PDE includes a 4 byte physical address which refers to a Page Table. Each Page Table has 1024 entries which either includes a physical address referring to a physical page of 4KB or includes an **invalid entry**. This was also discussed above in the processor's paging and Windows page table management section but repeated here for clearness sake. So why does Windows utilize logical address 0xC000000000 to amass the Page Tables and address 0xC0300000 to amass page directory? The prerequisite for storing the page tables in memory is that a rouge consumer mode application

Notes

should not be able to influence the page tables. Hence page tables should be in the kernel logical address space. Windows typically provides lower 2GB space to processes and reserves upper 2GB to kernel. But with a particular boot.ini switch /3GB, it permits user mode process to access lower 3GB memory. 0xC0000000 is the next address after 3GB and so I guess that is why it is selected for storing page directory and page tables. There are some other significant aspects to page tables and page directory layout in memory. To appreciate that, let us gaze at how page tables and page directory is laid out. To make it simple to appreciate, I have drawn page tables for a fake process with pertinent entries highlighted.

Caution Remember that every index entry is 4 bytes in size.

P_PT displays the physical address of a Page Table.

Index / Logical Address	0x0	0x1	.	0x80	.	0x300 (768)	0x34A (842)	.	.	0x400 (1023)
0xC0000000 0x6A078###	0x10480###	-				-				
0xC0001000 0x45045###	-	-					0x34005###			
0xC0002000										
-										
-										
0xC0100000										
-										
-										
0xC0300000 0x13453###	P_PT 0x6A078###	-		P_PT 0x45045###	-	PDB 0x13453###			-	
0xC0301000										
-										
-										
0xC03FF000										

PDB displays the physical address of page directory base of the subsequent process i.e. it shows the physical address subsequent to logical address 0xC0300000 for that process. This value is also accumulated in CR3.

Notes Windows can only make use of logical address to access any memory location counting page directory, thus to access page directory and page tables, it is essential to put some self referencing entry in page directory.

The physical address entries exposed above will be dissimilar for each process but each process will have its PDB entry stored at index 0x300 of Page directory.

We will execute logical to physical address translation on 4 dissimilar addresses to observe the significance of PDB entry, Page tables layout and how precisely the address translation works. The addresses which we will transform are 0x2034AC54, 0xC0000000, 0xC0300000, 0xC0300000 [0x300] i.e. 0xC030C00. First address is a usual user mode logical address for a process, the second address is the first logical address of first page table in logical address space, third address is logical address of page directory base and fourth address is logical address of a special entry as you will observe throughout translation. Assume CR3 refers to a physical address

Notes

0x13453###. As pointed previously, lower 12 bits are used to accumulate page protection information and other information required by an OS. These are out of the scope of our current discussion so I have shown them as ###. The upper 20 bits displays the Page Frame Number or PFN which is the physical address of a 4KB aligned page. The actual physical address corresponding to the PFN will be 0x13453000. Let us now perform the translation:

0x2034AC54

- 0x2034AC54 can be displayed as 0010000000 1101001010 110001010100
- The upper 10 bits which are 0010000000 provides the index into page directory. Transforming to hexadecimal, the upper 10 bits give a value of 0x080
- From CR3, we know the Page Directory is situated at physical address 0x13453000 and from discussion above we also know that Page Directory is situated at logical address
- 0xC0300000
- So 0xC0300000 [0x080] will give the address of page table which is P_PT. From the table above, we can observe that this address is displayed by page table at logical address 0xC00001000 (or physical address 0x45045000). Now we utilize next 10 bits i.e. 1101001010 (or 0x34A) to index into the page table.
- The address 0xC00001000 [0x34A] will provide us the physical address of a 4KB page which is 0x34005000 from the table above.
- The number displayed by lower 12 bits, which is 110001010100 (or 0xC54), is used to point to the actual byte on the 4KB page located at 0x34005000. The final physical address corresponding to 0x2034AC54 comes out to be 0x34005C54

I will leave the address translation of other 3 addresses as an exercise to the reader. Once you do that translation, you will appreciate why the PDB entry is stored at index 0x300 in the table above and it causes processor to consider page directory as a page table during address translation. Also this translation will provide you more information on why this specific layout was selected by windows designers.

Task Make distinction between logical address and physical address.

Self Assessment

Fill in the blanks:

14. Windows provides lower 2GB (or 3GB relying upon boot.ini switch) logical address space of a process to user mode and upper 2GB (or 1GB relying upon boot.ini switch) to Windows ..... .
15. The prerequisite for storing the page tables in ..... is that a rouge consumer mode application should not be able to influence the page tables.

2.5 Summary

- Memory management in Windows operating systems has developed into a affluent and classy architecture, competent of scaling from the minute embedded platforms all the way

Notes

up to the multi-terabyte NUMA configurations, taking full benefit of all capabilities of obtainable and upcoming hardware designs.

- Windows on 32 bit x86 systems can utilize up to 4GB of physical memory.
- The x86 processor separates the physical address space (or physical memory) in 4 KB pages.
- Every Page Directory Entry (or PDE) is 4 bytes in size and refers to a Page Table.
- In windows every process has its own Page Directory and Page Tables. Therefore windows assign 4 MB of this space per process.
- Windows give memory protection to all the processes in order that one process can't utilize other process's memory.
- Windows provides lower 2GB (or 3GB relying upon boot.ini switch) logical address space of a process to user mode and upper 2GB (or 1GB relying upon boot.ini switch) to Windows kernel.
- Each Page Table has 1024 entries which either includes a physical address referring to a physical page of 4KB or includes an invalid entry.

2.6 Keywords

Page Directory: The first dimension of X86 processor is called Page Directory.

Page Table: The second dimension of X86 processor is called Page Table

X86 processor: The X86 processor separates the physical address space (or physical memory) in 4 KB pages

2.7 Review Questions

1. What is Memory management in Windows operating systems? Explain.
2. Make distinction between logical memory address and physical address.
3. Illustrate the concept of Paging in X86 Processor.
4. Elucidate the two dimensional structure of X86 Processor.
5. What is page table? Illustrate the concept of page table with example.
6. Enlighten the concept of Windows Page Table Management.
7. How is the translation from logical to physical address performed? Discuss.
8. Illustrate how memory protection is performed.
9. How to execute logical to physical address translation on 4 dissimilar addresses? Illustrate.
10. The prerequisite for storing the page tables in memory is that a rouge consumer mode application should not be able to influence the page tables. Comment.

Answers: Self Assessment

1.	4GB	2.	Logical
3.	Physical	4.	Paging
5.	X86 Processor	6.	Page Directory

Notes

7. Page Table
8. Page Directory Entry
9. Physical Address
10. Valid
11. Invalid
12. Protection
13. User Mode
14. Kernel
15. Memory

2.8 Further Readings

Books

Brent E. Rector, *Win32 Programming,* Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

www.tenouk.com/visualcplusmfc/visualcplusmfc20.html

Notes

Unit 3: Windows Programming

CONTENTS

Objectives
Introduction
3.1 The Windows.h
3.2 WINMAIN Function
3.3 Creating the Programs Window
3.4 Messages and Adding a Message Loop
3.5 Creating a New Window Class
3.6 Message Processing Function WndProc()
3.7 Adding Custom Resource Data
3.7.1 Compiling the Resource Data
3.8 Summary
3.9 Keywords
3.10 Review Questions
3.11 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand windows.h
- Discuss WINMAIN function
- Recognize Creating the Programs Window
- Illustrate Messages and adding a message loop
- Discuss Creating a New Window Class
- Understand Message Processing Function WndProc()
- Discuss Adding and compiling custom resource data

Introduction

In this unit, you will understand the concepts of windows.h and WINMAIN function. You will study various concepts of windows programming such as Creating the Programs Window, Messages and adding a message loop, Creating a New Window Class, Message Processing Function WndProc(), etc.

3.1 The Windows.h

The Windows.h is a Windows-specific header file for the C programming language which comprises declarations for all of the functions in the Windows API, all the general macros used

Notes

by Windows programmers, and all the data types utilized by the different functions and subsystems. It defines a very huge number of Windows particular functions that can be utilized in C. The Win32 API can be added to a C programming project by comprising the <windows.h> header file and connecting to the appropriate libraries.

Caution To utilize functions in xxxx.dll, the program must be connected to xxxx.lib (or libxxxx.dll.a in MinGW).

Did u know? Some headers are not connected with a .dll but with a static library (e.g. scrnsave.h needs scrnsave.lib).

Self Assessment

Fill in the blanks:

1. The ..... is a Windows-specific header file for the C programming language which comprises declarations for all of the functions in the Windows API.
2. The Win32 API can be added to a C programming project by comprising the <windows.h> header file and connecting to the appropriate ..... .

3.2 WINMAIN Function

WINMAIN (also written as MAIN) is a user-defined function called by Windows to start execution of an application.

Syntax

```
FUNCTION {WINMAIN | MAIN} ( _
     BYVAL  hInstance    AS DWORD, _
     BYVAL  hPrevInst    AS DWORD, _
     BYVAL  lpszCmdLine  AS WSTRINGZ PTR, _
     BYVAL  nCmdShow     AS LONG ) AS LONG
```

The WINMAIN function is called by Windows when an executable application primarily loads and bstarts to run. It is frequently pointed to as the "entry point" for the application. When the execution of WINMAIN is accomplished, the application is estimated to be completed, and Windows releases the application memory back to the heap. WINMAIN obtains the following parameters:

1. ***hInstance:*** The executable's (EXE) *instance handle*. Each instance of a Windows application has an exclusive handle. It is used as a parameter to a number of Windows API functions which may want to differentiate among multiple instances of an application.
2. ***hPrevInst:*** It is not used by 32-bit Windows. It is there merely for compatibility with current 16-bit code, and always returns zero in 32-bit applications.
3. ***lpszCmdLine:*** A pointer to an nul-terminated string that comprises a command-line.

Notes

Notes Observe that the string passed in *lpszCmdLine* is not the similar as the string returned by the *GetCommandLine* API call. The string in *lpszCmdLine* comprises the command-line arguments only (like COMMAND$), but *GetCommandLine* returns the program name (counting path) followed by the arguments.

4. ***nCmdShow:*** It shows how to exhibit the application's main window.

Example: The calling application can identify %SW_NORMAL or %SW_MINIMIZE, etc. It is up to the programmer to respect this parameter, and to do so is suggested.

Return: The return value allocated to WINMAIN is optional, but by convention, the return value is derived from the *wParam&* parameter of a %WM_QUIT message.

Usually, a GUI-based application utilizes the WINMAIN function to generate the initial GUI application window, and then enters a message loop. This loop should end when a %WM_QUIT message is obtained, and the *wParam&* parameter of that message should be passed on as the return value for WINMAIN. If WINMAIN terminates before entering the message loop, WINMAIN should return zero.

Console applications may utilize the return value to situate an error level that can be passed back to the calling application, in the range 0 to 255 inclusive. Batch files may perform on the result through the IF [NOT] ERRORLEVEL batch command.

If the parameters passed to WINMAIN are not needed by the application itself, the PBMAIN function may be used in position of WINMAIN.

Restrictions: Pointers may not be passed BYREF, so the *lpszCmdLine* parameter of WINMAIN must be affirmed to be passed BYVAL.

Example:

```
#COMPILE EXE

FUNCTION WINMAIN(BYVAL hInst???, BYVAL hPrevInst???, BYVAL pCmdLine AS
WSTRINGZ PTR, BYVAL nCmdShow&) AS LONG

   ' more code here

   FUNCTION = 1

END FUNCTION
```

Task Illustrate the parameters *hPrevInst and lpszCmdLine..*

Self Assessment

Fill in the blanks:

3. ..... is a user-defined function called by Windows to start execution of an application.

4. The ..... value allocated to WINMAIN is optional, but by convention, the return value is derived from the *wParam&* parameter of a %WM_QUIT message.

Notes

3.3 Creating the Programs Window

There are two chief things you must perform in order to generate even the simplest window: you must create the middle point of the program, and you must tell the operating system how to react when the user does what.

Just like a C++ program always contains a **main()** function, a Win32 program requires a central function call **WinMain**. The syntax of that function is:

```
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine, int nCmdShow );
```

Dissimilar to the C++ **main()** function, the arguments of the **WinMain()** function are not optional. Your program will require them to converse with the operating system.

The first argument, *hInstance*, is a handle to the instance of the program you are writing.

The second argument, *hPrevInstance*, is used if your program had any earlier instance. If not, this argument can be unnoticed, which will always be the case.

The third argument, *lpCmdLine*, is a string that displays all items used on the command line to compile the application.

The last argument, *nCmdShow*, handles how the window you are building will be displayed.

An object that represents on your screen is known as a window. Since there can be different types of windows in your programs, your first accountability is to manage them, know where they are, what they are doing, why and when. The first control you must exercise on these dissimilar windows is to host them so that all windows of your program belong to an entity known as the main window. This main window is created by means of an object that can be called a class (strictly, a structure).

The Win32 library offers two classes for generating the main window and you can use any one of them. They are **WNDCLASS** and **WNDCLASSEX**. The second adds only a slight trait to the first. Thus, we will mostly use the **WNDCLASSEX** structure.

The **WNDCLASS** and the **WNDCLASSEX** classes are defined as below:

```
typedef struct _WNDCLASS {
   UINT       style;
   WNDPROC    lpfnWndProc;
   int        cbClsExtra;
   int        cbWndExtra;
   HINSTANCE  hInstance;
   HICON      hIcon;
   HCURSOR    hCursor;
   HBRUSH     hbrBackground;
   LPCTSTR    lpszMenuName;
   LPCTSTR    lpszClassName;
} WNDCLASS, *PW
```

```
typedef struct _WNDCLASSEX {
    UINT       cbSize;
    UINT       style;
    WNDPROC    lpfnWndProc;
    int        cbClsExtra;
    int        cbWndExtra;
    HINSTANCE  hInstance;
    HICON      hIcon;
    HCURSOR    hCursor;
    HBRUSH     hbrBackground;
    LPCTSTR    lpszMenuName;
    LPCTSTR    lpszClassName;
    HICON      hIconSm;
} WNDCLASSEX, *PWNDCLASSEX;
```

Caution To create a window, you must "fill out" this class, which means you must supply a value for each of its members so the operating system would identify what your program is anticipated to do.

Notes

The first thing you must do so as to create an application is to state a variable of either **WNDCLASS** or **WNDCLASSEX** type.

Example: Here is an example of a **WNDCLASSEX** variable:

```
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine, int nCmdShow)
{
      WNDCLASSEX WndClsEx;

     return 0;
}
```

Self Assessment

Fill in the blanks:

5. The argument, ... , is used if your program had any earlier instance.
6. The argument, ... , handles how the window you are building will be displayed.

3.4 Messages and Adding a Message Loop

Message loop is utilized to catch messages and pass them onto our WndProc function.

First we are required to state a MSG variable. This is used to hold the message that has been caught by the window.

MSG msg;

To discover out what the last message obtained was, we can use the GetMessage function. The parameters for the function are specified below.

LPMSG lpMsg - After the function call, the parameter passed onto here will enclose the message that was obtained.

HWND hwnd - This is used to state what window the message must be recovered for. If a NULL is passed, a message for any window will be retrieved.

UINT wMsgFilterMin & UINT wMsgFilterMax - This declares the range of messages to obtain. Each message has an integer value so you can identify exactly what messages you want to receive. If you pass 0 for both parameters, all messages are obtained.

Notes The return value is 0 if a WM_QUIT message has been obtained. If the return value is less than 0, some error has appeared. We only want to carry on if there was no error and if there is no quit message. Keep in mind that we caused a quit message to be sent if the window was closed.

When the GetMessage function is called, the function halts until a message is obtained. If you do not want this, you can make use of the PeekMessage function which doesn't stop, even if there was no message. This is typically used if you are busy computing something else. If you go in the course of the OpenGL or DirectX , you will use this.

Notes

```
while (GetMessage(&msg, NULL, 0, 0) > 0)
{
```

Functions are requisite to send the message onto the WndProc function.

First we make use of the TranslateMessage function. The translate message function translates virtual-key messages into character messages which is required. This function takes one parameter, being the MSG variable containing the message obtained.

```
TranslateMessage(&msg);
```

The next function we are required to use is the DispatchMessage function. This sends the message via the window procedure (WndProc). It also takes one MSGparameter.

```
DispatchMessage(&msg);
}
```

Thus we used the PostQuitMessage function and passed the value of 0. This could have been a dissimilar value we had passed so we cannot just return 0 as normal. We return the msg.wParam value which is the return code passed onto the PostQuitMessage function.

```
return (int)msg.wParam;
```

You should now be able to develop simple windows messages. If you execute the program, you will observe that the window will stay on the screen until you close the window. This will demolish the window and shut the program competently.

Task Make distinction between *LPMSG lpMsg* and *HWND hwnd.*

Self Assessment

Fill in the blanks:

7. Message loop is utilized to catch messages and pass them onto our ..... function.
8. ..... is used to state what window the message must be recovered for.
9. When the ..... function is called, the function halts until a message is obtained.

3.5 Creating a New Window Class

The **WNDLCLASS** and the **WNDCLASSEX** classes are utilized to initialize the application window class. To exhibit a window, that is, to provide the user an object to work with, you must generate a window object. This window is the object the user utilizes to interrelate with the computer.

To create a window, you can call either the **CreateWindow()** or the **CreateWindowEx()**function.

You can just call this function and state its arguments after you have registered the window class.

Example: Here is an example:

```
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine, int nCmdShow)
{
```

Notes

```
        WNDCLASSEX    WndCls;

        . . .

        RegisterClassEx(&WndClsEx);

        CreateWindow(.  .  .);
}
```

If you are preparing to use the window further in your application, you should recover the result of the **CreateWindow()** or the **CreateWindowEx()** function, which is a handle to the window that is being generated. To perform this, you can state an **HWND** variable and initialize it with the create function. This can be completed as follows:

```
INT  WINAPI  WinMain(HINSTANCE  hInstance,  HINSTANCE  hPrevInstance,
                  LPSTR  lpCmdLine,  int  nCmdShow)
{
        HWND   hWnd;
        WNDCLASSEX    WndClsEx;

        . . .

        RegisterClassEx(&WndClsEx);

        hWnd  =  CreateWindow(.  .  .);
}
```

Self Assessment

Fill in the blanks:

10. The **WNDLCLASS** and the ..... classes are utilized to initialize the application window class.
11. To create a window, you can call either the **CreateWindow()** or the ..... function.

3.6 Message Processing Function WndProc()

An application-defined function that processes messages sent to a window. The WNDPROC type defines a pointer to this callback function.

WindowProc is a placeholder for the application-defined function name.

Syntax

```
LRESULT  CALLBACK  WindowProc(
   __in   HWND  hwnd,
   __in   UINT  uMsg,
```

Notes

```
    __in   WPARAM wParam,
    __in   LPARAM lParam
);
```

Parameters

hwnd [in]

Type: HWND

A handle to the window.

uMsg [in]

Type: UINT

The message.

For lists of the system-provided messages, see System-Defined Messages.

wParam [in]

Type: WPARAM

Additional message information. The contents of this parameter depend on the value of the uMsg parameter.

lParam [in]

Type: LPARAM

Additional message information. The contents of this parameter depend on the value of the uMsg parameter.

Return value

Type: LRESULT

The return value is the result of the message processing and depends on the message sent.

If your application runs on a 32-bit version of Windows operating system, uncaught exceptions from the callback will be passed onto higher-level exception handlers of your application when available. The system then calls the unhandled exception filter to handle the exception prior to terminating the process. If the PCA is enabled, it will offer to fix the problem the next time you run the application.

However, if your application runs on a 64-bit version of Windows operating system or WOW64, you should be aware that a 64-bit operating system handles uncaught exceptions differently based on its 64-bit processor architecture, exception architecture, and calling convention. The following table summarizes all possible ways that a 64-bit Windows operating system or WOW64 handles uncaught exceptions.

Behavior type	How the system handles uncaught exceptions
1	The system suppresses any uncaught exceptions.
2	The system first terminates the process, and then the Program Compatibility Assistant (PCA) offers to fix it the next time you run the application. You can disable the PCA mitigation by adding a Compatibility section to the application manifest.
3	The system calls the exception filters but suppresses any uncaught exceptions when it leaves the callback scope, without invoking the associated handlers.

Notes

The following table shows how a 64-bit version of Windows operating system or WOW64 handles uncaught exceptions. Notice that behavior type 2 only applies to the 64-bit version of the Windows 7 operating system.

Operating System	WOW64	64-bit Windows
Windows XP	3	1
Windows Server 2003	3	1
Windows Vista	3	1
Windows Vista SP1	1	1
Windows 7	1	2

Notes On Windows 7 with SP1 (32-bit, 64-bit or WOW64), the system calls the unhandled exception filter to handle the exception prior to terminating the process. If the PCA is enabled, it will offer to fix the problem the next time you run the application.

If you need to handle exceptions in your application, you can use structured exception handling to do so. For more information on how to use structured exception handling, see Structured Exception Handling.

Requirements

Minimum supported client	Windows 2000 Professional
Minimum supported server	Windows 2000 Server
Header	Winuser.h (include Windows.h)

To permit Windows to converse with your application, we'll generate a great little function known as a Windows procedure. This most general name for this function is WndProc. This function MUST be formed and used to find out how your application will reaction to different events. The Windows procedure may also be called the event handler since it responds to Windows events! So let's have a speedy look at the prototype:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam);
```

This function is stated with a return type of LRESULT CALLBACK.

Did u know? The LRESULT type is used by Windows to state a long integer, and CALLBACK is a calling convention used with functions that are called by Windows.

The Windows Procedure is a function pointer, which permits you to call it whatever you want since the function's address will be allocated as a function pointer upon creation of the window class.

hwnd – It is significant only if you have numerous windows of the similar class open at one time. This is used to find out which window hwnd pointed to before deciding on an action.

Message - The actual message identifier that WndProc will be managing.

wParam and lParam – These are the Extensions of the message parameter. Used to provide more information and states that message cannot on its own.

Notes

Self Assessment

Fill in the blanks:

12. The Windows procedure may also be called the ..... since it responds to Windows events.
13. The Windows Procedure is a ..... pointer, which permits you to call it whatever you want since the function's address will be allocated as a function pointer upon creation of the window class.
14. ..... is a calling convention used with functions that are called by Windows.

3.7 Adding Custom Resource Data

You can generate a new custom or data resource by positioning the resource in a separate file using normal resource script (.rc) file syntax, and then involving that file by right-clicking your project in Solution Explorer and clicking **Resource Includes** on the shortcut menu.

To add a new custom or data resource:

1. Create a .rc file that comprises the custom or data resource. You can type custom data in a .rc file as null-terminated quoted strings, or as integers in decimal, hexadecimal, or octal format.
2. In **Solution Explorer**, right-click your project's .rc file, then click **Resource Includes** on the shortcut menu.
3. Then perform the process of compilation
4. Click **OK** to record your modifications.

3.7.1 Compiling the Resource Data

In the **Compile-Time Directives** box, type a **#include** statement that provides the name of the file including your custom resource.

Example: #include mydata.rc

Confirm that the syntax and spelling of what you type are accurate. The contents of the **Compile-Time Directives** box are inserted into the resource script file precisely as you typed them.

Another method to produce a custom resource is to import an external file as the custom resource.

Self Assessment

Fill in the blank:

15. In the **Compile-Time Directives** box, type a ..... statement that provides the name of the file including your custom resource.

3.8 Summary

- The Windows.h is a Windows-specific header file for the C programming language which comprises declarations for all of the functions in the Windows API, all the general macros

Notes

used by Windows programmers, and all the data types utilized by the different functions and subsystems.

- WINMAIN (also written as MAIN) is a user-defined function called by Windows to start execution of an application.
- The return value allocated to WINMAIN is optional, but by convention, the return value is derived from the *wParam&* parameter of a %WM_QUIT message.
- There are two chief things you must perform in order to generate even the simplest window: you must create the middle point of the program, and you must tell the operating system how to react when the user does what.
- The Win32 library offers two classes for generating the main window and you can use any one of them. They are **WNDCLASS** and **WNDCLASSEX**.
- Message loop is utilized to catch messages and pass them onto our WndProc function.
- To discover out what the last message obtained was, we can use the GetMessage function.
- The **WNDLCLASS** and the **WNDCLASSEX** classes are utilized to initialize the application window class.
- To permit Windows to converse with your application, we'll generate a great little function known as a Windows procedure.

3.9 Keywords

Return: The return value allocated to WINMAIN is optional, but by convention, the return value is derived from the *wParam&* parameter of a %WM_QUIT message.

Windows Procedure: To permit Windows to converse with your application, we'll generate a great little function known as a Windows procedure.

Windows.h: The Windows.h is a Windows-specific header file for the C programming language which comprises declarations for all of the functions in the Windows API.

WINMAIN: WINMAIN (also written as MAIN) is a user-defined function called by Windows to start execution of an application.

3.10 Review Questions

1. Illustrate the function of Windows.h.
2. What is WINMAIN? Write its syntax and illustrate the parameters.
3. Explain the use of return value allocated to WINMAIN.
4. Explain the process of creating the Programs Window.
5. Define the classes used in WNDCLASS and the WNDCLASSEX.
6. Illustrate the concept of WNDCLASSEX variable with example.
7. What are the parameters of GetMessage function? Illustrate.
8. How to create a new window class? Illustrate with example.
9. What is Message Processing Function WndProc()? Illustrate its parameters.
10. Describe the steps for adding and compiling custom resource data.

Notes

Answers: Self Assessment

1. Windows.h
2. Libraries
3. WINMAIN
4. Return
5. HPrevInstance
6. nCmdShow
7. WndProc
8. HWND Hwnd
9. GetMessage
10. WNDCLASSEX
11. CreateWindowEx()
12. Event Handler
13. Function
14. CALLBACK
15. #include

3.11 Further Readings

Books

Brent E. Rector, *Win32 Programming,* Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

www.flipcode.com/.../Introduction_To_Windows_Programming.sht.

Unit 4: Windows Controls

Notes

CONTENTS

Objectives

Introduction

4.1 Window

4.1.1 Types of Window Controls

4.2 The Create Windows() Function

4.3 Static Controls

4.4 C language Casts

4.5 Button Controls

4.5.1 Creating a Push Button

4.5.2 Characteristics of a Command Button

4.5.3 Enabling or Disabling a Button

4.5.4 The OK and Cancel Buttons

4.5.5 Processing

4.5.6 Button Notification Codes

4.6 List Boxes

4.6.1 Adding Items

4.6.2 Getting Data from the ListBox

4.7 Combo Boxes

4.7.1 Creating a Combo Box

4.7.2 Characteristics of a Combo Box

4.8 Scroll Bars

4.8.1 Types of Scroll Bars

4.8.2 Automatic Scroll Bars

4.8.3 Control-based Scroll Bars

4.9 Edit Controls

4.9.1 Edits with Numbers

4.10 Summary

4.11 Keywords

4.12 Review Questions

4.13 Further Readings

Notes

Objectives

After studying this unit, you will be able to:

- Understand types of controls
- Discuss the Create Windows() function
- Illustrate Static Controls and sending message to a Control
- Illustrate Button Controls and processing button control messages
- Understand List boxes, Combo boxes, Scroll bars, and Edit controls

Introduction

A control is considered as a child window that an application utilizes in combination with another window to allow user communication. Controls are most frequently used inside dialog boxes, but they can also be utilized in other windows. Controls inside dialog boxes offer the user with a way to type text, select options, and start actions. Controls in other windows offer a variety of services, like allowing the user select commands, view status, and view and edit text. This unit illustrates the controls offered by Windows and the programming elements utilized to create and influence them.

4.1 Window

Windows is a computer operating system from Microsoft that, in addition with some generally used business applications like Microsoft Word and Excel, has turn out to be a effective "standard" for individual users in most corporations and in most homes.

With the arrival of the Internet, Microsoft has relocated Windows as a type of "window to the world," and its attempts to take the lead in Web browsers have made Explorer the most well-liked browser. Microsoft's .NET initiative displays an effort to develop into industry-dominant in furnishing products and services that aid the use of remote application services on the Web.

4.1.1 Types of Window Controls

We will illustrate different types of windows controls in this unit such as:

- Static Control
- Button Controls
- List Boxes
- Combo Boxes
- Scroll Bars
- Edit Controls

Self Assessment

Fill in the blank:

1. A ..... is considered as a child window that an application utilizes in combination with another window to allow user communication.

Notes

4.2 The Create Windows() Function

Window controls are considered as predefined window classes i.e. you are not required to call the Register Class() function to generate a window class before the control.

Windows are usually generated using the "CreateWindow" function, even though there are a few other functions that are functional as well. Once a WNDCLASS has been registered, you can inform the system to make a window from that class by passing the class name (keep in mind that global string we defined?) to the CreateWindow function.

```
HWND CreateWindow(
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hWndParent,
    HMENU hMenu,
    HINSTANCE hInstance,
    LPVOID lpParam
);
```

The first parameter, "lpClassName" is the string connected with our window class. The "lpWindowName" parameter is the title that will be exhibited in the title bar of our window (if the window has a title bar).

"dwStyle" is a field that includes a number of bit-wise OR'd flags, that will control window creation.

Self Assessment

Fill in the blank:

2. Window controls are considered as predefined window classes i.e. you are not required to call the ..... function to generate a window class before the control.

4.3 Static Controls

We will illustrate here regarding the text static control. A text static control is like an edit control, but it does not obtain typed input from the user. A static control cannot be chosen and cannot obtain the keyboard focus. A static control is usually used as a label for other controls.

Example: If you have an edit control, you would usually use a static control on the left or above the edit control. This static control is a label and would have text that signifies the reason of the edit control.

The static control is of the system window class, STATIC.

Notes

The static control is considered as a child window. The following code exhibits a static control above an edit control:

```
#include
using namespace std;
LRESULT CALLBACK MainWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{

     switch (uMsg)
    {

         default:
                return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
     return 0;
}

int WINAPI WinMain(HINSTANCE hinstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{
     WNDCLASSEX wcx;
      wcx.cbSize = sizeof(wcx);
      wcx.style = CS_HREDRAW | CS_VREDRAW;
      wcx.lpfnWndProc = MainWndProc;
     wcx.cbClsExtra = 0;
     wcx.cbWndExtra = 0;
      wcx.hInstance = hinstance;
     wcx.hIcon = NULL;
      wcx.hCursor = LoadCursor(NULL, IDC_ARROW);
      wcx.hbrBackground = (HBRUSH)(COLOR_BACKGROUND+1);
     wcx.lpszMenuName =  NULL;
      wcx.lpszClassName = "MainWClass";
     wcx.hIconSm = NULL;

      RegisterClassEx(&wcx);

     HWND hwndMain;

      hwndMain = CreateWindowEx(0, "MainWClass", "Main Window",
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, NULL, NULL, hinstance, NULL);
```

Notes

```
     if (!hwndMain)
         return FALSE;

      ShowWindow(hwndMain, SW_SHOW);
      UpdateWindow(hwndMain);

     HWND hwndSt;
      hwndSt = CreateWindowEx(0, "STATIC", "First Name", WS_CHILD, 100, 100,
100, 20, hwndMain, (HMENU)1, hinstance, NULL);
      ShowWindow(hwndSt, SW_SHOW);
      UpdateWindow(hwndSt);
      HWND hwndEd = CreateWindowEx(0, "EDIT", NULL, WS_CHILD, 100, 122, 100,
20, hwndMain, (HMENU)2, hinstance, NULL);
      ShowWindow(hwndEd, SW_SHOW);
      UpdateWindow(hwndEd);

     MSG msg;
     BOOL bRet;

      while( (bRet = GetMessage( &msg, hwndMain, 0, 0 )) != 0)
         {
              if (bRet == -1)
             {
                    // handle the error and possibly exit the application
             }
             else
             {
                   TranslateMessage(&msg);
                   DispatchMessage(&msg);
             }
         }

      return msg.wParam;
}
```

You can utilize the following command at the command prompt to compile the code:

```
g++ winst.cpp -mwindows -o winst.exe
```

There are two controls in the code. Let us see at the static one. The class is, STATIC; this is the second argument of the CreateWindowEx function. The name of the static control is, "First Name"; this is the third argument of the function; this name occurs as the content display of the static control. You have the WS_CHILD style signifying that it is a child window. The rest of the arguments to the CreateWindowEx function are like those for the EDIT control. Keep in mind,

Notes

each control must have an exclusive integer identifier (at the tenth argument of the CreateWindowEx function).

If you open the application, winst.exe, by double-clicking on it, you would observe the 2 controls with the static control above the edit control. The static control contains the text "First Name" signifying that the user should type his first name in the edit control.

That is it for this part of the series. We stop here and carry on in the next part.

To turn up at any of the parts, just type the corresponding title below in the Search Box of this page and click Search

A Window Button Function and Macro

Windows Static Control

Static Control Styles

Sending Messages to a Control

All static controls are generated with the CreateWindowEx API function by changing the window styles. The API function CreateWindow can also be utilized. A new API function - SendMessage - is introduced. This function is utilized to converse with windows and controls.

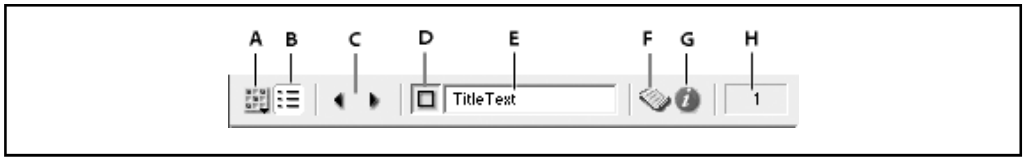
Self Assessment

Fill in the blanks:

3. A ..... control is like an edit control, but it does not obtain typed input from the user.
4. All static controls are generated with the ..... API function by changing the window styles.

4.4 C Language Casts

The controls at the top of the Cast window are the similar in both the List and Thumbnail views. Use the controls to modify the cast that occurs in the Cast window, the cast member selection, or the name of a cast member. You can also utilize them to shift cast members and to open a cast member's Script window or the Property inspector.



Cast Window Controls

A.

Cast

B.

Cast View Style

C.

Previous/Next Cast Member

Notes

D.

Drag Cast Member

E.

Cast Member Name

F.

Cast Member Script

G.

Cast Member Properties

H.

Cast Member Number

Change the Cast Displayed in the Current Cast Window

Do one of the following:

- Click the Cast button and select a cast from the menu.
- Click a tabbed panel to make it active.
- Press Control+Alt (Windows) or Command+Option (Mac) followed by the Right Arrow key or Left Arrow key to move from tab to tab.

Open a cast in a New Cast Window

Click the Cast button and select a cast from the context menu.

Select the Previous or Next Cast Member

Click the Previous Cast Member or Next Cast Member button.

Move a selected Cast Member to a New Position in the Cast Window (Thumbnail view) or to the Stage

Drag the Drag Cast Member button to the desired position in the Cast window or to the Stage. This procedure is useful when the selected cast member has scrolled out of view.

Enter a Cast Member Name

Select a cast member and enter the name in the Cast Member Name text box.

Edit a Cast Member Script

Select a cast member and click the Cast Member Script button.

View Cast Member Properties

1. Select a cast member.

Notes

2. Do one of the following:
   - Click the Cast Member Properties button.
   - Right-click (Windows) or Control-click (Mac), and select Cast Member Properties from the context menu.
   - Select Window > Property Inspector. The Property inspector displays only those properties associated with the selected cast member.

View the Cast Member Number

See the Cast Member Number field in the upper-right corner of the Cast window.

Self Assessment

Fill in the blank:

5. The controls at the top of the Cast window are the similar in both the List and ..... views.

4.5 Button Controls

A Button is a Windows control utilized to initiate an action. From the user's point of view, a button is functional when clicked, in which case the user places the mouse on it and presses one of the mouse's buttons.

There are numerous types of buttons. The most general and regularly utilized is a rectangular object that the user can simply identify. In some programming surroundings, this classic type is known as a command button. There are other controls that can serve as click controls and initiate the similar behavior as if a button were clicked.

From the programmer's point of view, a button requires a host, like a dialog box.

4.5.1 Creating a Push Button

To diagrammatically add a button to a dialog box, in the Toolbox, click Button and click the required location on the dialog box. By default, when you visually generate a dialog box, Microsoft Visual C++ adds two buttons: OK and Cancel. If you don't require these buttons, click one and press Delete.

The most accepted button utilized in Windows applications is a rectangular control that exhibits a word or a short sentence that directs the consumer to access, dismiss, or initiate an action or suite of actions. In Microsoft Visual C++ applications, this control is executed by means of the Button control from the Toolbox window.

In the MFC, a button depends on the **CButton** class, which is obtained from **CWnd**. Thus, to programmatically obtain a button, you can create a pointer to **CButton** and initialize it by means of the **new** operator.

4.5.2 Characteristics of a Command Button

Similar to every Windows control, a button is identified by its **ID**entifier. Since a button is a control, by convention, its identifier's name begins with IDC (the C stands for Control).

Notes

If you are required to programmatically access the properties of a control without using a connected variable, you may have to call the **CWnd::GetDlgItem()** member function. It occurs in two versions as follows:

```
CWnd* GetDlgItem(int nID) const;
void CWnd::GetDlgItem(int nID, HWND* phWnd) const;
```

When calling this member function, the first version permits you to allocate a **CWnd** (or derived class) to it. The second version returns a handle to the window passed as pointer.

Caution In both cases, you must pass the identifier of the control that you would like to access.

When using the first version, if the control is not a **CWnd** object, you must cast it to its local class. Then you can influence the property (or properties) of your option.

Example: Here is an example that accesses a button and modifies its caption:

```
BOOL CDialog5Dlg::OnInitDialog()
{
      CDialog::OnInitDialog();

      SetIcon(m_hIcon, TRUE); // Set big icon
      SetIcon(m_hIcon, FALSE); // Set small icon

      // TODO: Add extra initialization here

      CButton *btnWhatElse = reinterpret_cast<CButton
*>(GetDlgItem(IDC_BUTTON3));

      return TRUE; // return TRUE unless you set the focus to a control
}
```

The second version needs a pointer to a child window that you would like to access.

4.5.3 Enabling or Disabling a Button

For the user to be capable to utilize a control like clicking a button, the control must permit it. This trait of Windows objects is handled by the **CWnd::EnableWindow()** member function. Its syntax is:

```
BOOL EnableWindow(BOOL bEnable = TRUE);
```

This member function is utilized to enable or disable a control.

Did u know? The default value of the argument *bEnable* is set to TRUE, which would exhibit a control.

To disable a control, set the argument to FALSE.

Notes

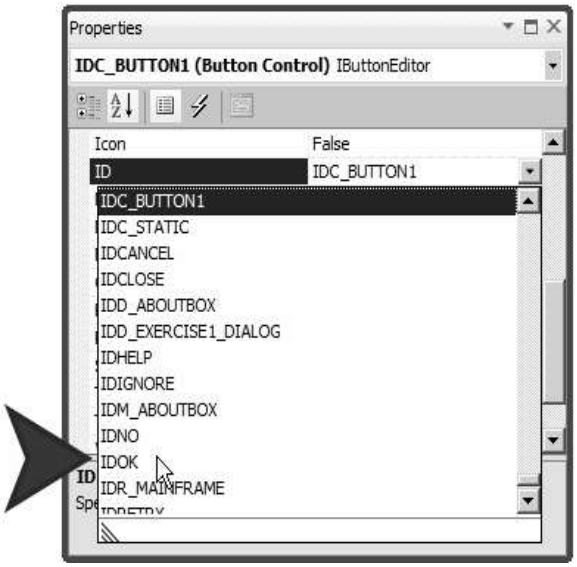
Example: Here is an example:

```
void CDialog5Dlg::OnLetItBe()
{
      // TODO: Add your control notification handler code here
      m_Submit.EnableWindow(FALSE);
}
```

4.5.4 The OK and Cancel Buttons

The most well-liked button captions in dialog boxes are OK and Cancel. The OK caption is set for a dialog box that notifies the user of an error, a mediator situation, or an acknowledgement of an action that was accomplished on the dialog that hosts the button. Microsoft Visual C++ makes it simple to add an OK button since in Windows applications, the OK object has a particular meaning.

To use an OK button, add a button to a form and, from the ID combo box, choose the **IDOK**identifier:



What makes this constant unique is that the MFC library identifies that, when the user clicks it, if the dialog box is modal, the user is recognizing the situation. If this dialog box was called from another window by means of the **DoModal()** member function, you can locate out if the user had clicked OK and then you can take further action. Thus when the user clicks OK, the dialog box calls the **OnOK()** member function. Its syntax is:

```
virtual void OnOK();
```

Even though it appears like a simple member function (and it is), the **OnOK()** member function carries the constant value **IDOK** that you can utilize as a return value of the **DoModal()**member function. Thus, in one step, you can utilize the **DoModal()** member function to exhibit a modal dialog box and discover whether the user clicked OK.

When a dialog box is prepared with an OK button, you should permit the user to press Enter and execute the OK clicking. This is taken care of by setting the Default Button property to True or checked.

Notes

The Cancel caption is functional on a button whose parent (the dialog box) would ask a question or request a follow-up action from the user. A Cancel button is also simple to make by just adding a button to a dialog box and choosing **IDCANCEL** as its identifier in the ID combo box. Setting a button's identifier to **IDCANCEL** also permits the user to press Esc to dismiss the dialog box.

The scenarios illustrated for the OK and the Cancel buttons are made probable only if the compiler is able to make sure or validate the changes completed on a dialog box. To make this validation probable, in your class, you must overload the **CWnd::DoDataExchange()** member function. Its syntax is:

```
virtual void DoDataExchange(CDataExchange* pDX);
```

This member function is utilized internally by the application (the framework) to discover if data on a dialog box has been changed as the object was displayed. This member function does two things: It makes sure the exchange of data among controls and it authenticates the values of those controls. In realism, it does not intrinsically perform data validation, meaning it would not permit or disallow value on a control. Rather, the compiler uses it to generate a table of the controls on the dialog box, their variables and values, permitting other controls to refer to it for data exchange and validation. If you want to discover the data a user would have typed or chosen in a control, you would have to write the essential code.

4.5.5 Processing

Button Control Messages

The most usual action users carry out on a button is to click it. When a user does this, the button sends a **BN_CLICKED** message. In some but rare situations, you may also ask the user to double-click a button. In general, you will take care of most message handling when the user clicks a button. There are other messages that you can manage through a button.

To close a dialog box, you can utilize the Win32 API's **PostQuitMessage()** function. Its syntax is:

```
VOID PostQuitMessage(int nExitCode);
```

This function takes one argument, which is an integer. The argument could be set to approximately any integer value even though it should be **WM_QUIT**.

Example: Here is an example:

```
void CDialog5Dlg::OnBtnClose()
{
      // TODO: Add your control notification handler code here
      PostQuitMessage(125);
}
```

Even though the MFC library offers enough messages related with the various controls, in some situation you will need use a message that is not necessarily connected with the control. In such a case, you can call the **CWnd::SendMessage()** member function. Its syntax is:

LRESULT SendMessage(UINT *message*, WPARAM *wParam* = 0, LPARAM *lParam* = 0);

The first argument of this member function can be a Win32 message or constant.

Notes

Example: Examples would be **WM_CLOSE** or **WM_ACTIVATE**.

The *wParam* and *lParam* arguments can be supplementary (Win32) messages.

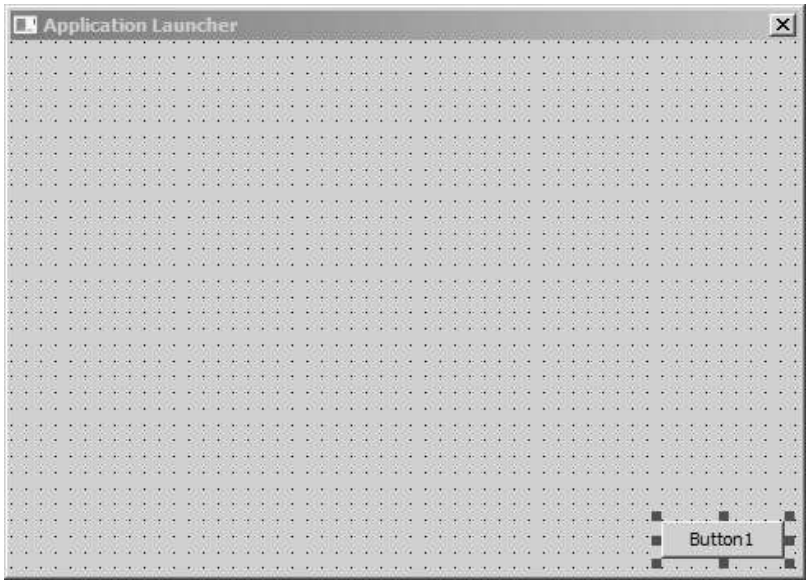
The **WinExec()** function is used to execute an application. Its syntax is:

```
UINT WinExec(LPCSTR lpCmdLine, UINT uCmdShow);
```

The *lpCmdLine* argument states the name or path of the application you want to exhibit. The uCmdShow states how the application should be exhibited. It utilizes the same values as the CWnd::ShowWindow**()** member function.

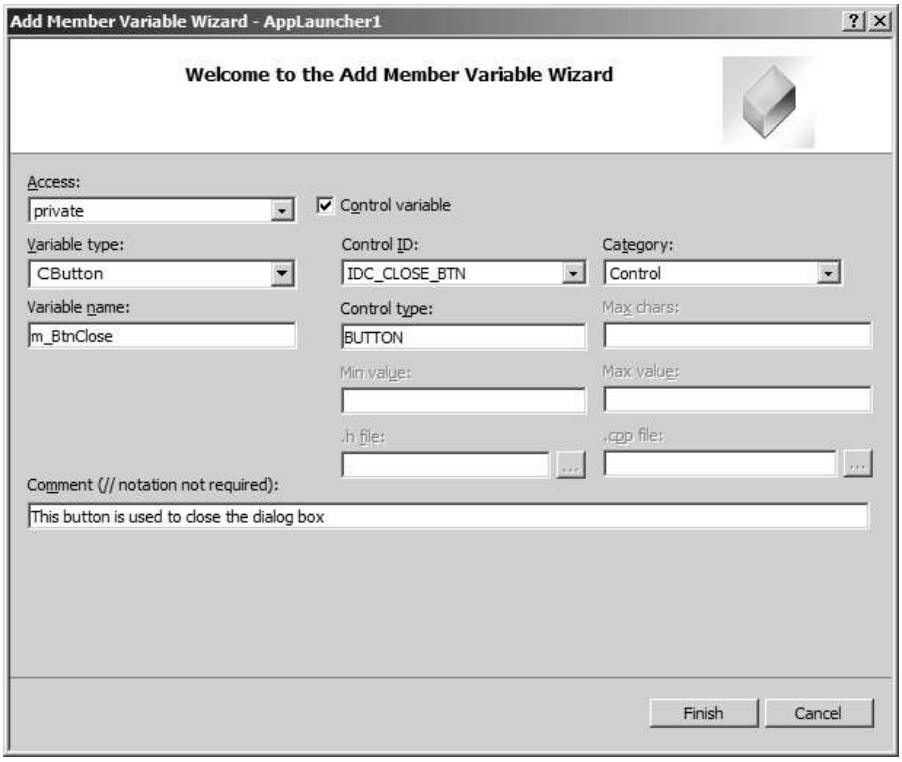
Practical Learning: Using Buttons

1. To generate a new application, on the main menu, click File -> New Project...
2. Click MFC Application and set the name to **AppLauncher1**
3. Click OK
4. In the first page of the MFC Application Wizard, click Next
5. In the second page, click Dialog Based
6. Click Next
7. In the third page, set the Title Name to Application Launcher
8. Click Next
9. Click Finish
10. On the dialog, click the TODO line and press Delete
11. As the OK button is selected, press Delete
12. As the Cancel button is selected, press Delete
13. In the Toolbox, click the Button control and click the lower section of the dialog box



Notes

14. In the Properties window, click ID and type **IDC_CLOSE_BTN**
15. On the dialog box, right-click the button and click Add Variable...
16. In the Access combo box, select private
17. In the Variable Name, type **m_BtnClose**



18. Click Finish
19. In the Class View, enlarge the project. In the upper part of the Class View, click CAppLauncher1
20. In the lower part of the Class View, double-click OnInitDialog()
21. Set the caption of the button to "Close" as below:

```
BOOL CAppLauncher1Dlg::OnInitDialog()
{
CDialogEx::OnInitDialog();

      // Add "About..." menu item to system menu.

      // IDM_ABOUTBOX must be in the system command range.
      ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
      ASSERT(IDM_ABOUTBOX < 0xF000);

      CMenu* pSysMenu = GetSystemMenu(FALSE);
     if (pSysMenu != NULL)
```

Notes

```
{
   BOOL bNameValid;
   CString strAboutMenu;
   bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
   ASSERT(bNameValid);
   if (!strAboutMenu.IsEmpty())
   {
         pSysMenu->AppendMenu(MF_SEPARATOR);
         pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
   }
}

// Set the icon for this dialog.  The framework does this
automatically
   //  when the application's main window is not a dialog
   SetIcon(m_hIcon, TRUE);            // Set big icon
   SetIcon(m_hIcon, FALSE);           // Set small icon

   // TODO: Add extra initialization here
   m_BtnClose.SetWindowText(L"Close");

   return TRUE;  // return TRUE  unless you set the focus to a control
}
```

22. To execute the application, press F5
23. To close the dialog box, click its System Close button , , or
24. Display the dialog box
25. Right-click the button and click Add Event Handler
26. In the Event Handler Wizard, make sure the Message Type is set to BN_CLICKED Ensure the Class List is set to CAppLauncher1Dlg. Click the Add And Edit button
27. Execute the OnClick event as follows:

```
void CAppLauncher1Dlg::OnBnClickedCloseBtn()
{
    // TODO: Add your control notification handler code here
    PostQuitMessage(WM_QUIT);
}
```

28. To execute the application, press F5
29. To close the dialog box, click the Close button and return to your programming environment

4.5.6 Button Notification Codes

Notes

Notifications

Topic	Contents
BCN_DROPDOWN	Sent when the user clicks a drop down arrow on a button. The parent window of the control obtains this notification code in the form of a WM_NOTIFY message.
BCN_HOTITEMCHANGE	Informs the button control owner that the mouse is entering or leaving the client area of the button control. The button control sends this notification code in the form of a WM_NOTIFY message.
BN_CLICKED	Sent when the user clicks a button. The parent window of the button obtains the BN_CLICKED notification code through the WM_COMMANDmessage.
BN_DBLCLK	Sent when the user double-clicks a button. This notification code is sent automatically for BS_USERBUTTON,BS_RADIOBUTTON, and BS_OWNERDRAW buttons. Other button types send BN_DBLCLK only if they have the BS_NOTIFY style. The parent window of the button obtains the BN_DBLCLK notification code via the WM_COMMAND message.
BN_DISABLE	Sent when a button is disabled. This notification code is offered only for compatibility with 16-bit versions of Windows earlier than version 3.0. Applications should use the BS_OWNERDRAW button style and the DRAWITEMSTRUCT structure for this task. The parent window of the button obtains the BN_DISABLE notification code via the WM_COMMANDmessage.
BN_DOUBLECLICKED	Sent when the user double-clicks a button. This notification code is sent automatically for BS_USERBUTTON,BS_RADIOBUTTON, and BS_OWNERDRAW buttons. Other button types send BN_DOUBLECLICKED only if they have the BS_NOTIFY style. The parent window of the button obtains the BN_DOUBLECLICKED notification code via the WM_COMMAND message.
BN_HILITE	Sent when the user selects a button. This notification code is offered only for compatibility with 16-bit versions of Windows earlier than version 3.0. Applications should use the BS_OWNERDRAW button style and the DRAWITEMSTRUCT structure for this task. The parent window of the button obtains the BN_HILITE notification code via the WM_COMMAND message.
BN_KILLFOCUS	Sent when a button loses the keyboard focus. The button must have the BS_NOTIFY style to send this notification code. The parent window of the button receives the BN_KILLFOCUS notification code via the WM_COMMANDmessage.
BN_PAINT	Sent when a button should be painted. This notification code is offered only for compatibility with 16-bit versions of Windows earlier than version 3.0. Applications should use the BS_OWNERDRAW button style and the DRAWITEMSTRUCT structure for this task. The parent window of the button obtains the BN_PAINT notification code through the WM_COMMAND message.
BN_PUSHED	Sent when the push state of a button is set to pushed. This notification code is offered only for compatibility with 16-bit versions of Windows earlier than version 3.0. Applications should

Contd...

Notes

	use the BS_OWNERDRAW button style and the DRAWITEMSTRUCT structure for this task. The parent window of the button obtains the BN_PUSHED notification code through the WM_COMMANDmessage.
BN_SETFOCUS	Sent when a button obtains the keyboard focus. The button must have the BS_NOTIFY style to send this notification code. The parent window of the button obtains the BN_SETFOCUS notification code through the WM_COMMAND message.
BN_UNHILITE	Sent when the highlight should be removed from a button. This notification code is provided only for compatibility with 16-bit versions of Windows earlier than version 3.0. Applications should utilize the BS_OWNERDRAW button style and the DRAWITEMSTRUCT structure for this task. The parent window of the button obtains the BN_UNHILITE notification code via the WM_COMMAND message.
BN_UNPUSHED	Sent when the push state of a button is set to unpushed. This notification code is provided only for compatibility with 16-bit versions of Windows earlier than version 3.0. Applications should use the BS_OWNERDRAW button style and the DRAWITEMSTRUCT structure for this task. The parent window of the button obtains the BN_UNPUSHED notification code via the WM_COMMAND message.
NM_CUSTOMDRAW (button)	Informs the parent window of a button control regarding custom draw operations on the button. The button control sends this notification code in the form of a WM_NOTIFY message.
WM_CTLCOLORBTN	The WM_CTLCOLORBTN message is sent to the parent window of a button before drawing the button. The parent window can modify the button's text and background colors. However, only owner-drawn buttons react to the parent window processing this message.

Task Make distinction between BN_CLICKED and BN_DOUBLECLICKED.

Self Assessment

Fill in the blanks:

6. From the user's point of view, a ..... is functional when clicked, in which case the user places the mouse on it and presses one of the mouse's buttons.
7. To close a dialog box, you can utilize the Win32 API's ..... function.
8. A notification code ..... is send when the button is disabled.

4.6 List Boxes

List box is the last standard control which is considered as a handy tool.

4.6.1 Adding Items

The first thing you'll desire to do with a listbox is add items to it.

```
int index = SendDlgItemMessage(hwnd, IDC_LIST, LB_ADDSTRING, 0, (LPARAM)"Hi
there!");
```

Notes

Notes As you can observe, this is a quite simple task. If the listbox has the LBS_SORT style, the new item will be added in alphabetical order, or else it will just be added to the end of the list.

This message returns the index of the new item either manner and we can utilize this to perform other tasks on the item, like relating some data with it. Generally this will be things like a pointer to a struct including more information, or maybe an ID that you will use to identify the item, it's up to you.

```
SendDlgItemMessage(hwnd, IDC_LIST, LB_SETITEMDATA, (WPARAM)index,
(LPARAM)nTimes);
```

4.6.2 Getting Data from the ListBox

Now that we recognize the selection has changed, or at the request of the user, we require to obtain the selection from the listbox and do something functional with it.

Example: In this example I've used a multi-selection list box, so obtaining the list of chosen items is a little trickier. If it were a single selection listbox, than you could just send LB_GETCURSEL to retrieve the item index.

First we require to obtain the number of selected items, so that we can assign a buffer to save the indexes in.

```
HWND hList = GetDlgItem(hwnd, IDC_LIST);
int count = SendMessage(hList, LB_GETSELCOUNT, 0, 0);
```

Then we allocate a buffer based on the number of items, and send LB_GETSELITEMS to fill in the array.

```
int *buf = GlobalAlloc(GPTR, sizeof(int) * count);
SendMessage(hList, LB_GETSELITEMS, (WPARAM)count, (LPARAM)buf);

// ... Do stuff with indexes

GlobalFree(buf);
```

```
In this example, buf[0] is the first index, and so on up to buf[count - 1].
```

One of the things you would likely want to do with this list of indexes, is recover the data connected with each item, and perform some processing with it. This is just as uncomplicated as setting the data was originally, we just send another message.

```
int data = SendMessage(hList, LB_GETITEMDATA, (WPARAM)index, 0);
```

If the data was some other type of value (anything that is 32-bits) you could just cast to the suitable type.

Example: For example if you accumulated HBITMAPs instead of ints...

HBITMAP hData = (HBITMAP)SendMessage(hList, LB_GETITEMDATA, (WPARAM)index, 0);

Notes

Self Assessment

Fill in the blank:

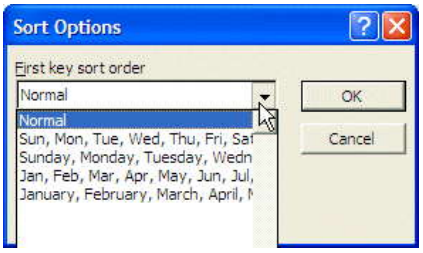
9. If the data was some other type of value (anything that is 32-bits) you could just ..... to the suitable type.

4.7 Combo Boxes

A combo box is a Windows control prepared of two sections. There are two major types of combo boxes: drop down and simple. Each is prepared of two sections.

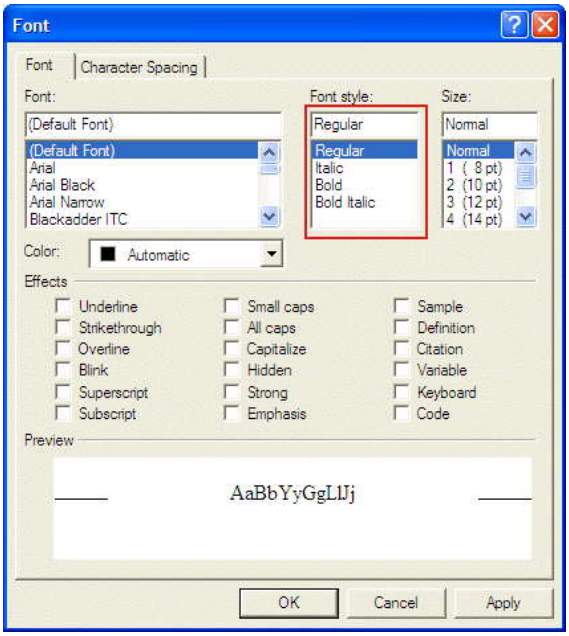
The most generally used combo box is called drop down. On the left side, it is prepared of an edit box. On the right side, it is equipped with a down-pointing arrow:

To utilize it, the user must click the arrow. This opens a list:



After finding the desired item in the list, the user can click it. The item clicked turns out to be the new one exhibiting in the edit part of the control. If the user doesn't locate the desired item in the list, he or she can click the down-pointing arrow or press Esc. This hides the list and the control displays as before. The user can also exhibit the list by providing focus to the control and then pressing Alt + down arrow key.

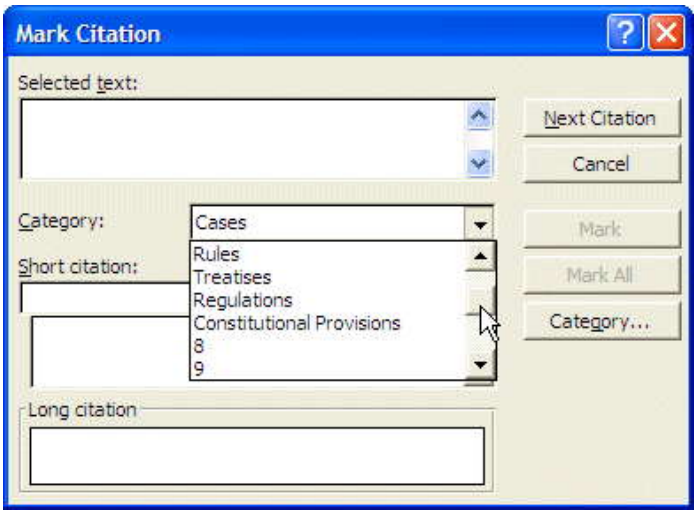
The second general type of combo box is pointed to as simple. This type is also made of two sections but, rather than a down-pointing arrow used to exhibit the list, it displays its list all the time:

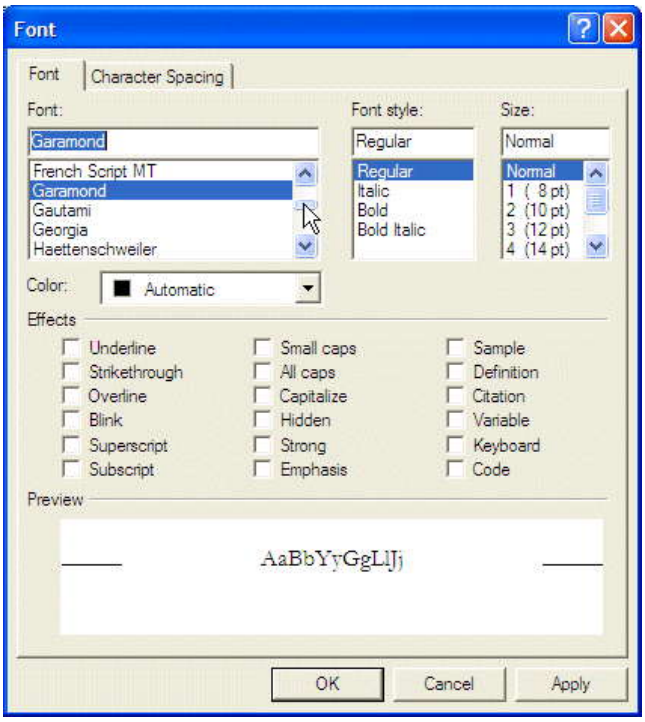


Notes

This time, to choose an item, the user can just locate it in the list and click it.

In both types of combo boxes, if the list is too long for the assigned space, when it exhibits, the list part is equipped with a vertical scroll bar. This permits the user to navigate up and down in the list to locate the desired item:





4.7.1 Creating a Combo Box

Three are two main manners in which you can create a combo box. You can write code or utilize a script. To create a combo box with code, you can first create a Windows class that defines an **HWND** handle and executes the allocations of a combo box.

Notes

The simplest manner to create a combo box is via a resource script. The syntax utilized to create the control in a script is:

COMBOBOX *id, x, y, width, height [, style [, extended-style]]*

Caution You must state **COMBOBOX** as the class of this control.

The *id* is the number used to recognize the control in a resource header file

The *x* measure is its horizontal location with respect to the control's origin, which is located in the top left corner of the window that is hosting the combo box

The *y* factor is the distance from control's origin, which is located in the top left corner of the window that is hosting the combo box, to the top-left side of the combo box

The *width* and the *height* specify the dimensions of the combo box

The optional *style* and the *extended-style* factors are used to configure and organize the behavior of the combo box.

To create a combo box follow the steps as below:

1. To create an identifier for the combo box, open the resource header file and modify it as follows:

```
#define IDD_CONTROLSDLG 101
#define IDD_SIZE_CBO    102
```

2. To create the combo box, open the resource script and modify it as follows:

```
#include "resource.h"

IDD_CONTROLSDLG DIALOG 260, 200, 180, 120
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Windows Controls"
FONT 8, "MS Shell Dlg"
BEGIN
   DEFPUSHBUTTON   "Close", IDCANCEL, 120, 100, 50, 14
   COMBOBOX        IDD_SIZE_CBO, 40, 8, 90, 80
    END
```

3. Test the application

4. Click Close to dismiss the dialog box.

Notes

4.7.2 Characteristics of a Combo Box

Windows Styles of a Combo Box

Similar to all the other windows, to generate a combo box programmatically, you can call the **CreateWindow()** or the **CreateWindowEx()** function. The syntax used is:

```
HWND CreateWindow( "COMBOBOX",
                   "Default String",
                   style,
                   x,
                   y,
                   width,
                   height,
                   parent,
                   menu,
                   instance,
                   Optional arameter );
```

```
HWND CreateWindowEx( Extended Style,
                     "COMBOBOX",
                     "Default String",
                     style,
                     x,
                     y,
                     width,
                     height,
                     parent,
                     menu,
                     instance,
                     Optional Parameter );
```

The first argument of the **CreateWindow()** or the second argument of the **CreateWindowEx()** functions must be **COMBOBOX** passed as a string.

The second argument of the **CreateWindow()** or the third argument of the **CreateWindowEx()** functions states a string that would display in the edit part of the combo box when the control occurs. If the control is generated with certain styles we will review here, this string would not come out even if you state it. You can also omit it and pass the argument as **NULL** or "" since there are other ways you can set the default string.

Similar to every other Windows control, a combo box appearance and behavior are managed by a set of properties called styles.

Did u know? The main properties of a combo box are those handled by the operating system and shared by all controls.

You can utilize them to set the visibility, availability, and parenthood, etc., of the combo box. If you create a combo box by means of a resource script, since you would contain it in a **DIALOG** section of the script, the dialog box is automatically made its parent. Or else, to specify that the combo box is hosted by another control, get the handle of the host and pass it as the parent parameter. You must also set or add the **WS_CHILD** bit value to the style parameter. If you want the combo box to occur when its parent comes up, add the **WS_VISIBLE** style using the bitwise | operator.

If you want the combo box to obtain focus as a result of the user pressing the Tab key, add the **WS_TABSTOP** style.

The location of a combo box is stated by the *x* and *y* parameters whose values are d on the origin, located in the top-left corner or the dialog box or the window that is hosting the combo box.

Programmatically Creating a Combo Box

1. To programmatically create a combo box, alter the Exercise.cpp file as follows:

```
#include <windows.h>
```

Notes

```
#ifdef __BORLANDC__
#pragma argsused
#endif

#include "resource.h"
//————————————————————————————————————
HWND hWnd;
HWND hWndComboBox;
HINSTANCE hInst;
LRESULT CALLBACK DlgProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM
     lParam);
//————————————————————————————————————
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
     LPSTR lpCmdLine, int nCmdShow )
{
     hInst = hInstance;

DialogBox(hInstance, MAKEINTRESOURCE(IDD_CONTROLSDLG),
hWnd, reinterpret_cast<DLGPROC>(DlgProc));

return 0;
}
//————————————————————————————————————
LRESULT CALLBACK DlgProc(HWND hWndDlg, UINT Msg, WPARAM wParam, LPARAM
     lParam)
{
     switch(Msg)
       {
          case WM_INITDIALOG:
          hWndComboBox = CreateWindow("COMBOBOX",
               NULL,
   WS_CHILD | WS_VISIBLE | WS_TABSTOP,
     60, 62, 136, 60,
            hWndDlg,
          NULL,
          hInst,
           NULL);
```

Notes

```
if( !hWndComboBox )
  {
    MessageBox(hWndDlg,
"Could not create the combo box",
"Failed Control Creation",
       MB_OK);
   return FALSE;
    }
     return TRUE;
  case WM_COMMAND:
     switch(wParam)
  {
  case IDCANCEL:
  EndDialog(hWndDlg, 0);
            }
     break;
  }

  return FALSE;
     }
//————————————————————————————————————
```

2. Test the application
3. Click the Close button to dismiss it

Self Assessment

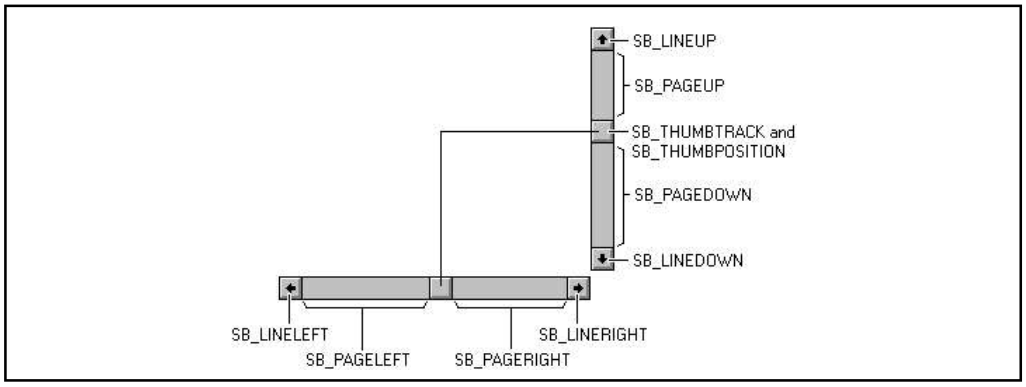
Fill in the blanks:

10. If the user doesn't locate the desired item in the list, he or she can click the ..... arrow or press Esc.
11. The first argument of the **CreateWindow()** or the second argument of the ..... functions must be **COMBOBOX** passed as a string.

4.8 Scroll Bars

A scroll bar is an object that permits the user to navigate either left and right or up and down, either on a document or on a section of the window. A scroll bar occurs as a long bar with a (small) button at each end. Among these buttons, there is a moveable bar known as a thumb. To scroll, the user can click one of the buttons or grab the thumb and drag it:

Notes



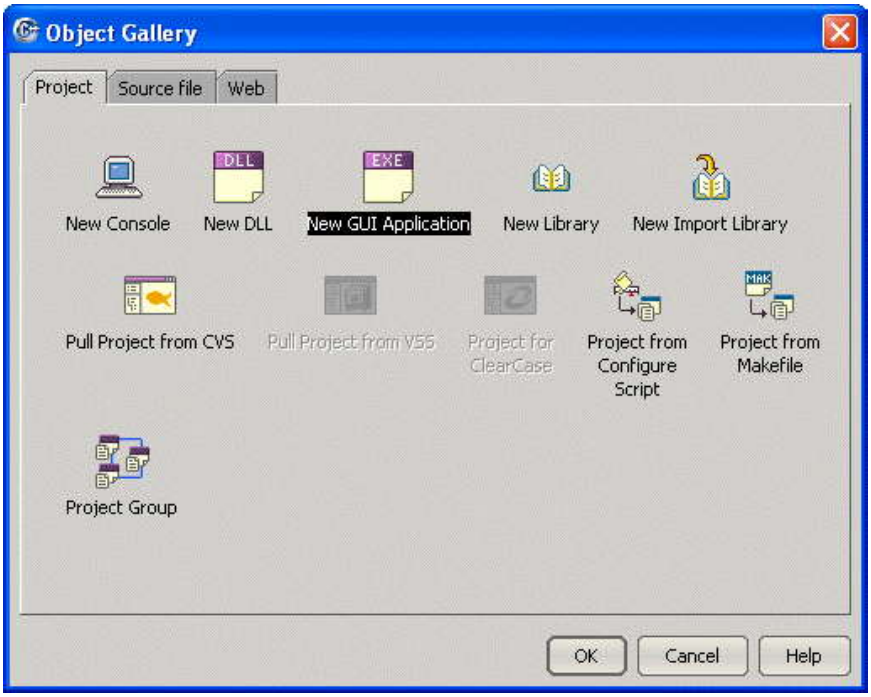
4.8.1 Types of Scroll Bars

There are two types of scroll bars: vertical or horizontal. A vertical scroll bar permits the user to navigate up and down on a document or a section of a window. A horizontal scroll bar permits the user to navigate left and right on a document or a section of a window.

As far as Microsoft Windows is considered, there are two groups of scroll bars: automatic and control-based.

To create scroll bars, follow the steps as below:

1. Since Borland C++BuilderX is free, we are going to utilize it. Begin Borland C++ Builder X and, on the main menu, click File -> New...

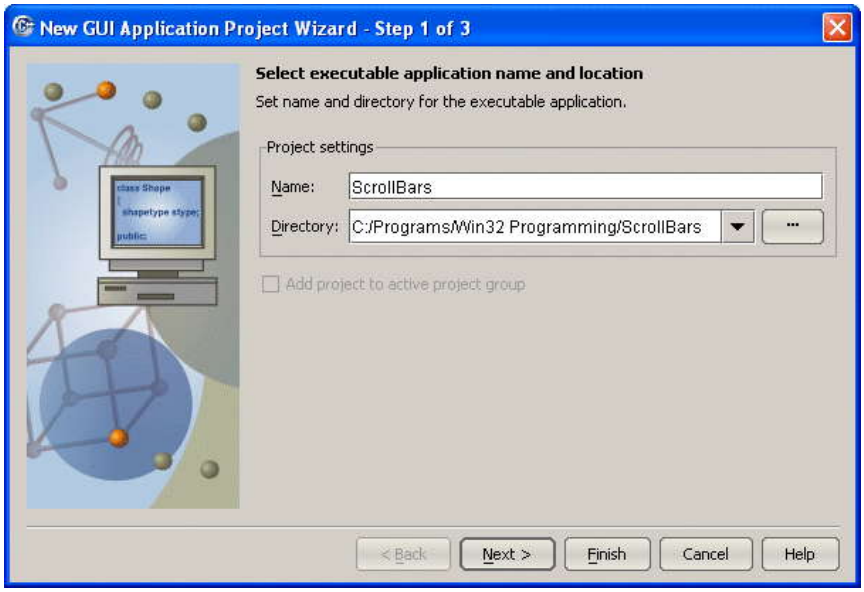


2. In the Object Gallery dialog box, click New GUI Application and click OK

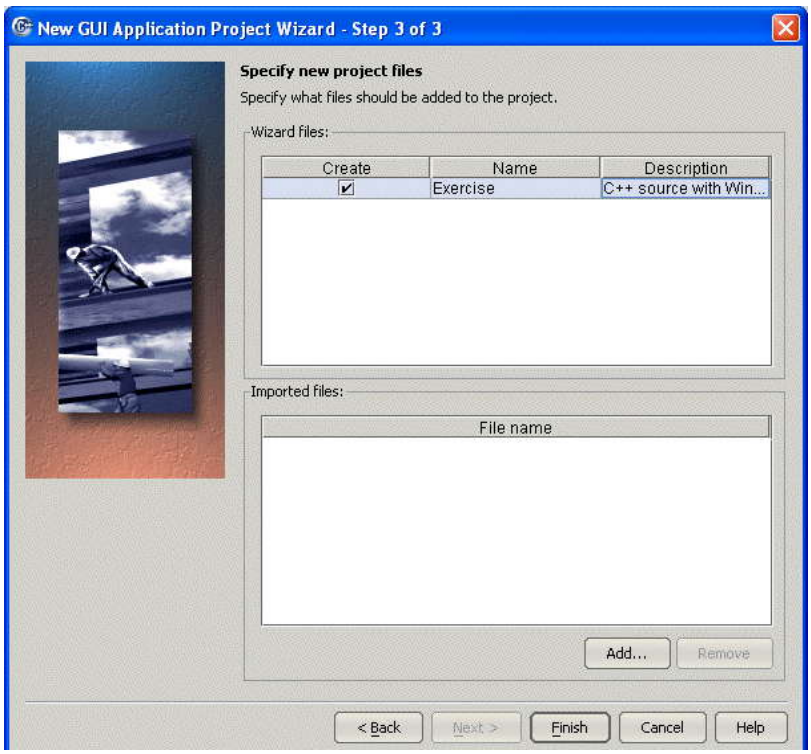
3. In the New GUI Application Project Wizard - Step 1 of 3, in the Directory edit box of the Project Settings section, type the path you desire. Or else, type **C:\Programs\Win32 Programming**

Notes

4. In the Name edit box, type **ScrollBars**



5. Click Next
6. In the New GUI Application Project Wizard - Step 2 of 3, accept the defaults and click Next
7. In the New GUI Application Project Wizard - Step 3 of 3, click the check box under Create
8. Choose Untitled under the Name column header. Type Exercise to swap the name and press Tab



Notes

9. Click Finish
10. In the left frame, double-click Exercise.cpp and modify the file to the following:

```
#include <windows.h>
#ifdef __BORLANDC__
#pragma argsused
#endif

const char *ClsName = "CtrlExos";
const char *WndName = "Controls Examples";
HINSTANCE hInst;

LRESULT CALLBACK WndProcedure(HWND hWnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam);

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
{
        MSG        Msg;
        HWND       hWnd;
        WNDCLASSEX WndClsEx;

        hInst = hInstance;

        // Create the application window
        WndClsEx.cbSize        = sizeof(WNDCLASSEX);
        WndClsEx.style         = CS_HREDRAW | CS_VREDRAW;
        WndClsEx.lpfnWndProc   = WndProcedure;
        WndClsEx.cbClsExtra    = 0;
        WndClsEx.cbWndExtra    = 0;
        WndClsEx.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
        WndClsEx.hCursor       = LoadCursor(NULL, IDC_ARROW);
       WndClsEx.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
        WndClsEx.lpszMenuName  = NULL;
        WndClsEx.lpszClassName = ClsName;
        WndClsEx.hInstance     = hInst;
        WndClsEx.hIconSm       = LoadIcon(NULL, IDI_APPLICATION);
```

Notes

```
        // Register the application
        RegisterClassEx(&WndClsEx);

        // Create the window object
        hWnd = CreateWindow(ClsName,
                            WndName,
                            WS_OVERLAPPEDWINDOW,
                            CW_USEDEFAULT,
                            CW_USEDEFAULT,
                            CW_USEDEFAULT,
                            CW_USEDEFAULT,
                            NULL,
                            NULL,
                            hInst,
                            NULL);

        // Find out if the window was created successfully
        if( !hWnd ) // If the window was not created,
                return 0; // stop the application

        // Display the window to the user
        ShowWindow(hWnd, SW_SHOWNORMAL);
        UpdateWindow(hWnd);

        // Decode and treat the messages
        // as long as the application is running
        while( GetMessage(&Msg, NULL, 0, 0) )
        {
            TranslateMessage(&Msg);
            DispatchMessage(&Msg);
        }

        return Msg.wParam;
}

LRESULT CALLBACK WndProcedure(HWND hWnd, UINT Msg,
                              WPARAM wParam, LPARAM lParam)
{
```

Notes

```
        switch(Msg)
        {
        case WM_CREATE:
           // There is nothing significant to do at this time
            return 0;

        case WM_DESTROY:
            // If the user has finished, then close the window
            PostQuitMessage(WM_QUIT);
            break;
        default:
            // Process the left-over messages
            return DefWindowProc(hWnd, Msg, wParam, lParam);
        }
        // If something was not done, let it go
        return 0;
    }
```

11. Press F9 to test the application

Task Make distinction between vertical scroll bar and horizontal scroll bar.

4.8.2 Automatic Scroll Bars

Some controls require a scroll bar to competently implement their functionality. The main example is the edit control, which is used to exhibit text. On that control, when the text is too long, the user is required to be able to scroll down and up to access the document fully. In the similar manner, if the text is too wide, the user is required to be able to scroll left and right to view the whole document.

When creating a text-based document or window, you can simply ask that one or both scroll bars be added. Certainly, an edit control must be able to handle multiple lines of text. This is taken care of by adding the **ES_MULTILINE** flag to its styles. Then:

- To add a vertical scroll bar to the window, add the **WS_VSCROLL** flag to the *Style argument* of the **CreateWindow()** or the **CreateWindowEx()** function.
- To add a horizontal scroll bar to the window, add the **WS_HSCROLL** flag to the *Style argument* of the **CreateWindow()** or the **CreateWindowEx()** function.
- To make the vertical scroll bar appear when necessary, that is, when the document is too long, add the **ES_AUTOVSCROLL** style.
- To make the horizontal scroll bar appear as soon as at least one line of the document is too wide, add the **ES_AUTOVSCROLL** style.

Obviously, you can use only one, two, three or all four styles.

Automatically Handling Scroll Bars

Notes

1. To create a small editor with its scroll bars, modify the procedure as follows:

```
LRESULT CALLBACK WndProcedure(HWND hWnd, UINT Msg,
                          WPARAM wParam, LPARAM lParam)
{
    static HWND hWndEdit;

    switch(Msg)
    {
    case WM_CREATE:

       hWndEdit = CreateWindow("EDIT", // We are creating an Edit
     control
                             NULL,   // Leave the control empty
                             WS_CHILD | WS_VISIBLE | WS_HSCROLL |
                                 WS_VSCROLL | ES_LEFT |
     ES_MULTILINE |
                                 ES_AUTOHSCROLL | ES_AUTOVSCROLL,
                             0, 0, 0, 0, // Let the WM_SIZE
     message below take care of the size
                              hWnd,
                             0,
                              hInst,
                              NULL);

        return 0;

    case WM_SETFOCUS:
        SetFocus(hWndEdit);
        return 0;

    case WM_SIZE:
       MoveWindow(hWndEdit, 0, 0, LOWORD(lParam), HIWORD(lParam),
     TRUE);
        return 0;

    case WM_DESTROY:
        // If the user has finished, then close the window
```

Notes

```
            PostQuitMessage(WM_QUIT);
            break;
        default:
            // Process the left-over messages
            return DefWindowProc(hWnd, Msg, wParam, lParam);
        }
        // If something was not done, let it go
        return 0;
    }
```

2. Test the application

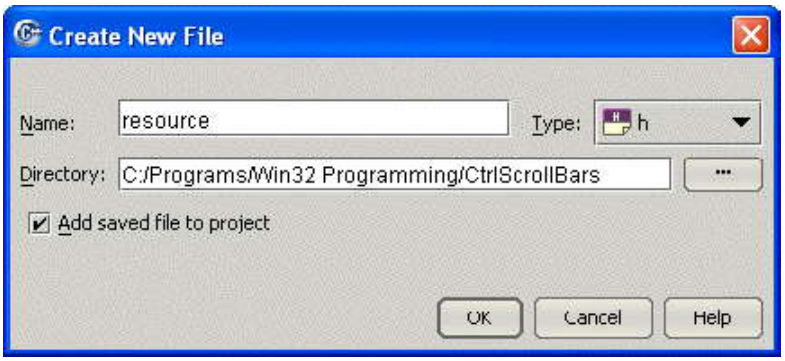
4.8.3 Control-based Scroll Bars

Microsoft Windows offers another type of scroll bar. Treated as its own control, a scroll bar is created like any other window and can be positioned anywhere on its host.

To create a scroll bar as a Windows control, call the **CreateWindow()** or the CreateWindowEx**()** functions and specify the class name as **SCROLLBAR**.

Using Scroll Bar Controls

1. Begin a new GUI Application and name it **CtrlScrollBars**
2. Create its accompanying file as Exercise.cpp
3. To create a resource header file, on the main menu, click File -> New File...
4. In the Create New File dialog box, in the Name, type **resource**
5. In the Type combo box, select h

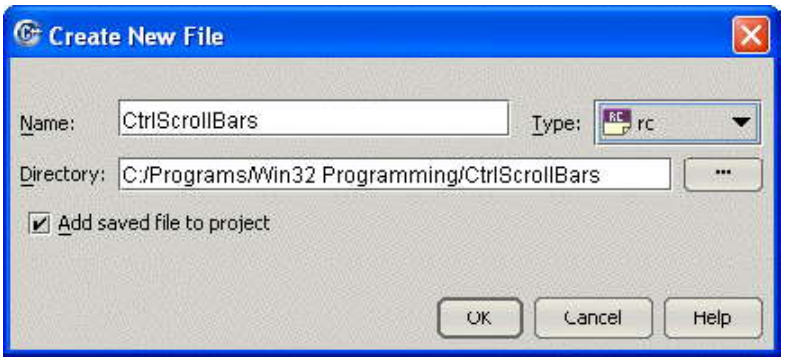


6. Click OK
7. In the file, type:
8. #define IDD_CONTROLS_DLG 101

   #define IDC_CLOSE_BTN 1000
9. To create a resource script, on the main menu, click File -> New File...

Notes

10. In the Create New File dialog box, in the Name, type **CtrlScrollBars**
11. In the Type combo box, select rc



12. Click OK
13. In the file, type:

```
#include "resource.h"

IDD_CONTROLS_DLG DIALOG DISCARDABLE  200, 150, 235, 151
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Windows Controls"
FONT 8, "MS Sans Serif"
BEGIN
     PUSHBUTTON
   "&Close",IDC_CLOSE_BTN,178,7,50,14
END
```

14. Display the Exercise.cpp file and change it as follows:

```
#include <windows.h>
#ifdef __BORLANDC__
   #pragma argsused
#endif

#include "resource.h"

//----------------------------------------
HWND hWnd;
HINSTANCE hInst;
LRESULT CALLBACK DlgProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam);
//----------------------------------------
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow )
{
     hInst = hInstance;
```

Notes

```
        DialogBox(hInst, MAKEINTRESOURCE(IDD_CONTROLS_DLG),
                    hWnd, reinterpret_cast<DLGPROC>(DlgProc));

   return 0;
}
//-----------------------------------------
LRESULT CALLBACK DlgProc(HWND hWndDlg, UINT Msg,
                         WPARAM wParam, LPARAM lParam)
{
          switch(Msg)
         {
          case WM_INITDIALOG:
                  return TRUE;

          case WM_COMMAND:
                   switch(wParam)
                  {
                     case IDC_CLOSE_BTN:
                          EndDialog(hWndDlg, 0);
                         return TRUE;
                  }
                  break;

          case WM_CLOSE:
                    PostQuitMessage(WM_QUIT);
                  break;
         }

          return FALSE;
}
//-----------------------------------------
```

15. Test the application

Self Assessment

Fill in the blanks:

12. A ..... is an object that permits the user to navigate either left and right or up and down, either on a document or on a section of the window.
13. To ..... a scroll bar as a Windows control, call the **CreateWindow()** or the CreateWindowEx() functions and specify the class name as **SCROLLBAR**.

Notes

4.9 Edit Controls

One of the most generally used controls in the windows environment, the EDIT control, is used to permit the user to enter, modify, copy, etc... text. Windows Notepad is little more than a plain old window with a big edit control inside it.

Example: Here is the code used to interface with the edit control in this example:

```
SetDlgItemText(hwnd, IDC_TEXT, "This is a string");
```

That's all it takes to modify the text included in the control (this can be used for pretty much any control that has a text value connected with it, STATICs, BUTTONs and so on).

Retrieving the text *from* the control is simple as well, though a little more work than setting it...

```
 int len = GetWindowTextLength(GetDlgItem(hwnd, IDC_TEXT));
if(len > 0)
{
    int i;
    char* buf;

     buf = (char*)GlobalAlloc(GPTR, len + 1);
     GetDlgItemText(hwnd, IDC_TEXT, buf, len + 1);

    //... do stuff with text ...

     GlobalFree((HANDLE)buf);
}
```

Initially, we need to assign some memory to store the string in, it won't just return us a pointer to the string previously in memory. So as to do this, we first require to know how much memory to assign. There isn't a GetDlgItemTextLength(), but there is a GetWindowTextLength(), so all we require to do is obtain the handle to the control by means of GetDlgItem().

Now that we have the length, we can assign some memory. Here I've added a verify to observe if there is any text to start with, as most likely you don't want to be working with an empty string... at times you might, but that's up to you. Presuming that there *is* something there to function with, we call GlobalAlloc() to assign some memory. GlobalAlloc() as I've used it here is equivalent to calloc(), if you're used to DOS/UNIX coding. It assigns some memory, initializes it's contents to 0 and returns a pointer to that memory. There are dissimilar flags you can pass as the first parameter to make it behave differently for different purposes, but this is the only manner we will be using it.

Notes Observe that we added 1 to the length in two positions, what's up with that? Well, GetWindowTextLength() returns the number of characters of text the control includes NOT INCLUDING the null terminator. This signifies that if we were to assign a string

Contd...

Notes

without adding 1, the text would fit, but the null terminator would overflow the memory block, perhaps corrupting other data, causing an access violation, or any number of other bad things. You must be careful when dealing with string sizes in windows, some APIs and messages expect text lengths to comprise the null and others don't, always read the docs comprehensively.

Lastly we can call GetDlgItemText() to retrieve the contents of the control into the memory buffer that we've just assigned. This call expects the size of the buffer INCLUDING the null terminator. The return value, which we unnoticed here, is the number of characters copied, NOT involving the null terminator.... fun eh? :)

After we're all done by means of the text (which we'll get to in a moment), we are required to free up the memory that we assigned so that it doesn't leak out and drip down onto the CPU and short circuit your computer. To achieve this, we simply call GlobalFree() and pass in our pointer.

You may be or become conscious of a second set of APIs named LocalAlloc(), LocalFree(), etc... which are legacy APIs from 16-bit windows. In Win32, the Local* and Global* memory functions are identical.

4.9.1 Edits with Numbers

Entering text is all well and okay, but what if you would like the user to enter in a number? This is a pretty general task, and luckily there is an API to make this easier, which takes care of all the memory allocation, in addition to converting the string to an integer value.

```
BOOL bSuccess;
int nTimes = GetDlgItemInt(hwnd, IDC_NUMBER, &bSuccess, FALSE);
```

GetDlgItemInt() functions much like GetDlgItemText(), except that instead of copying the string to a buffer, it transforms it internally into an integer and returns the value to you. The third parameter is optional, and takes a pointer to a BOOL. As the function returns 0 on failure, there is no way to tell just from that whether or not the function failed or the user just entered 0. If you are fine with a value of 0 in the event of an error, then feel free to disregard this parameter.

Another functional trait is the ES_NUMBER style for edit controls, which permits only the characters 0 through 9 to be entered. This is very handy if you only want positive integers, otherwise it's not much good, as you can't enter any other characters, involving - (minus) . (decimal) or , (comma).

Self Assessment

Fill in the blanks:

14. The ..... control is used to permit the user to enter, modify, copy, etc... text.
15. Initially, we need to assign some memory to store the ..... in, it won't just return us a pointer to the string previously in memory.

4.10 Summary

- Window controls are considered as predefined window classes i.e. you are not required to call the RegisterClass() function to generate a window class before the control.
- A text static control is like an edit control, but it does not obtain typed input from the user.

Notes

- From the user's point of view, a button is functional when clicked, in which case the user places the mouse on it and presses one of the mouse's buttons.
- The most usual action users carry out on a button is to click it. When a user does this, the button sends a BN_CLICKED message.
- The most generally used combo box is called drop down. On the left side, it is prepared of an edit box. On the right side, it is equipped with a down-pointing arrow:
- A scroll bar is an object that permits the user to navigate either left and right or up and down, either on a document or on a section of the window.
- To create a scroll bar as a Windows control, call the CreateWindow() or the CreateWindowEx() functions and specify the class name as SCROLLBAR.
- One of the most generally used controls in the windows environment, the EDIT control, is used to permit the user to enter, modify, copy, etc... text.

4.11 Keywords

BN_CLICKED: The most usual action users carry out on a button is to click it. When a user does this, the button sends a BN_CLICKED message.

Scroll Bar: A scroll bar is an object that permits the user to navigate either left and right or up and down, either on a document or on a section of the window.

Static Control: A static control is like an edit control, but it does not obtain typed input from the user.

4.12 Review Questions

1. Illustrate the function that is used for creating windows.
2. What is static control? Discuss the process of sending Messages to a control.
3. What are button controls? Illustrate the steps for creating a push button.
4. Explain the concept of processing button control messages. Illustrate with example.
5. What are different types of Button Notification Codes? Explain.
6. How to add items and obtain data from the list box?
7. What is a combo box? Illustrate the steps for creating a combo box. Also illustrate it programmatically.
8. What is a scroll bar? Make distinction between Automatic Scroll Bars and control-based scroll bar.
9. Illustrate step by step the process of using Scroll Bar Controls.
10. Illustrate the concept of edit controls with example.

Answers: Self Assessment

1. control
2. Register Class ()
3. text static
4. CreateWindowEx

Notes

5. Thumbnail
6. button
7. PostQuitMessage()
8. BN_DISABLE
9. cast
10. down-pointing
11. CreateWindowEx()
12. scroll bar
13. create
14. EDIT
15. string

4.13 Further Readings

Books

Brent E. Rector, *Win32 Programming,* Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

www.winprog.org/tutorial/controls.html

Unit 5: Memory Management (I)

Notes

CONTENTS

Objectives
Introduction
5.1 Local vs Global Memory
5.1.1 Global Memory
5.1.2 Local Memory
5.2 Local Memory Blocks
5.3 Using Fixed Memory Blocks
5.3.1 Fixed Size Memory Management Configuration
5.4 Changing the Size of a Memory Block
5.5 Using LocalReAlloc()
5.5.1 Parameters
5.5.2 Return Values
5.6 Discardable Memory Blocks
5.7 Summary
5.8 Keywords
5.9 Review Questions
5.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand local vs global memory
- Discuss local memory blocks
- Understand using fixed memory blocks
- Discuss changing the size of the memory block
- Illustrate using LocalReAlloc()
- Discuss discardable memory blocks

Introduction

Memory management under Win32 from the application's standpoint is pretty simple and uncomplicated. Each process possesses a 4 GB memory address space. The memory model used is known as a flat memory model. In this model, all segment registers (or selectors) point to the similar beginning address and the offset is 32-bit so an application can use memory at any point in its own address space without the requirement to modify the value of selectors. This shortens memory management a lot. There's no "near" or "far" pointer any longer.

Notes

5.1 Local vs Global Memory

5.1.1 Global Memory

Global memory exists in device memory and device memory is used through 32-, 64-, or 128-byte memory transactions. These memory transactions must be normally aligned: Only the 32, 64-, or 128-byte segments of device memory that are aligned to their size (i.e. whose first address is a multiple of their size) can be read or written by memory communications. When a warp implements an instruction that uses global memory, it coalesces the memory accesses of the threads inside the warp into one or more of these memory transactions relying on the size of the word accessed by every thread and the distribution of the memory addresses across the threads. Usually, the more transactions are essential, the more unused words are relocated as well as the words accessed by the threads, decreasing the instruction throughput therefore.

Example: If a 32-byte memory transaction is produced for each thread's 4-byte access, throughput is divided by 8. How many transactions are essential and how throughput is eventually affected differs with the compute potential of the device?

For devices of compute capability 1.0 and 1.1, the necessities on the distribution of the addresses across the threads to obtain any coalescing at all are very severe. They are much more comfortable for devices of superior compute capabilities. For devices of compute capability 2.0, the memory communications are cached, so data locality is exploited to decrease impact on throughput. To make the most of global memory throughput, it is thus significant to maximize coalescing by:

- Following the most optimal access patterns.
- Using data types that fulfill the size and alignment prerequisite as illustrated below.
- Padding data in some cases, for instance, when accessing a two-dimensional array as illustrated below.

Size and Alignment Requirement

Global memory instructions assist reading or writing words of size equal to 1, 2, 4, 8, or 16 bytes. Any access (through a variable or a pointer) to data existing in global memory compiles to a single global memory instruction if and only if the size of the data type is 1, 2, 4, 8, or 16 bytes and the data is logically allocated (i.e. its address is multiple of that size). If this size and alignment prerequisite is not fulfilled, the access compiles to various instructions with interleaved access patterns that stop these instructions from completely coalescing. It is as a result suggested to use types that fulfill this prerequisite for data that exists in global memory.

Did u know? The allocation prerequisite is automatically fulfilled for built-in types.

For structures, the size and alignment necessities can be enforced by the compiler using the alignment specifiers

```
__attribute__ ((aligned(8)))
or
__attribute__ ((aligned(16))) , such as
struct{floata;floatb;}__attribute__((aligned(8)));
```

Notes

```
or
struct{floata;floatb;floatc;}__attribute__((aligned(16)));
```

Any address of a variable existing in global memory or returned by one of the memory allotment routines from the driver or runtime API is always allocated to at least 256 bytes. Reading non-naturally aligned 8-byte or 16-byte words generates incorrect results (off by a few words).

Caution Special care must be taken to preserve alignment of the beginning address of any value or array of values of these types.

A classic case where this might be easily unnoticed is when using some tradition global memory allocation scheme, whereby the allocations of numerous arrays

(with multiple calls to

```
cudaMalloc()
or
cuMemAlloc()
```

) is substituted by the allocation of a single large block of memory partitioned into multiple arrays, in which case the starting address of every array is offset from the block's starting address.

Two-Dimensional Arrays

A general global memory access pattern is when every thread of index (tx,ty) accesses the following address to access one element of a 2D array of width ,situated at address BaseAddress of type type*:

BaseAddress + width * ty + tx

For these accesses to be completely coalesced, both the breadth of the thread block and the breadth of the array must be a numerous of the warp size (or only half the warp size for devices of compute capability 1.x). Particularly, this signifies that an array whose width is not a multiple of this size will be accessed much more competently if it is actually assigned with a width rounded up to the closest multiple of this size and its rows padded consequently.

5.1.2 Local Memory

Local memory accesses only happen for some automatic variables. Automatic variables that the compiler is probable to position in local memory are Arrays for which it cannot find out that they are indexed with constant quantities,

- Large structures or arrays that would use too much register space,
- Any variable if the kernel utilizes more registers than obtainable (this is also called register spilling).

Notes Observe that some mathematical functions have completion paths that might access local memory. The local memory space exists in device memory, so local memory accesses have similar high latency and low bandwidth as global memory accesses and are subject to the similar necessities for memory coalescing.

Notes

Local memory is however managed such that consecutive 32-bit words are used by consecutive thread IDs. Accesses are as a result fully coalesced as long as all threads in a warp access the similar relative address (e.g. similar index in an array variable, similar member in a structure variable). On devices of compute capability 2.0, local memory accesses are at all times cached in L1 and L2 in the similar manner as global memory accesses

Self Assessment

Fill in the blanks:

1. ..... memory exists in device memory and device memory is used through 32, 64–, or 128-byte memory transactions.
2. For structures, the size and alignment necessities can be enforced by the compiler using the alignment .....
3. ..... memory accesses only happen for some automatic variables.

5.2 Local Memory Blocks

Under Win16, there are two major groups of memory API functions: Global and Local. Global-type API calls deal with memory assigned in other segments therefore they're "far" memory functions. Local-type API calls deal with the local heap of the process so they're "near" memory functions. Under Win32, these two types are indistinguishable. Whether you call GlobalAlloc or LocalAlloc, you get the similar result.

Steps in assigning and using memory are as below:

1. Assign a block of memory by calling LocalAlloc. This function returns a handle to the requested memory block.
2. "Lock" the memory block by calling LocalLock. This function accepts a handle to the memory block and returns a pointer to the memory block.
3. You can utilize the pointer to read or write memory.
4. "Unlock" the memory block by calling LocalUnlock . This function cancels the pointer to the memory block.
5. Free the memory block by calling LocalFree. This function accepts the handle to the memory block.

You can also replace "Local" by "Global" like GloballAlloc, GlobalLock, etc.

The above method can be further simplified by means of a flag in GlobalAlloc call, GMEM_FIXED. If you use this flag, the return value from Global/LocalAlloc will be the pointer to the assigned memory block, not the memory block handle. You don't have to call Global/LocalLock and you can pass the pointer to Global/LocalFree without calling Global/LocalUnlock first.

Self Assessment

Fill in the blanks:

4. Local-type API calls deal with the local ..... of the process so they're "near" memory functions.
5. ..... function returns a handle to the requested memory block.

Notes

5.3 Using Fixed Memory Blocks

Fixed size memory blocks allocation algorithm has been established to permit technique of memory allocation in static time, separately to number of allocated blocks. This method is extensively used in real-time applications.

First plan that should be executed is to arrange the memory by means of stFixedMemInit function. Just after it can be utilized stFixedMemAlloc to assign memory blocks and stFixedMemFree to release assigned memory blocks in a provided memory.

All of the allocated blocks have a fixed size, defined throughout memory initialization. The size is allocated to value stated in AR_MEMORY_ALIGNMENT constant, defined in architecture specific files. Block addresses are always allocated not relatively to the NULL but to address specified throughout memory initialization.

Functions do not sets the previous error code on breakdown. Access to the fixed size memory is never harmonized. If it is compulsory it should be implemented.

If the fixed size memory management module will not be utilized, it can be expelled from compilation by setting ST_USE_FIXMEM for 0.

Now we illustrate some examples of using fixed size memory management module.

Example: Fixed size memory pool initialization

The example below displays how to initialize memory pool for fixed size memory blocks allocation.

```
#include <stdio.h>
#include "ST_API.h"

UINT8 MemoryPool[1024];

int main(void)
{
   PVOID Block;

   /* Initialization */
   arInit();
   stInit();

   /* Initialize memory pool for fixed-size memory blocks allocation.
      Size of the memory block is set to 16 bytes */
   if(!stFixedMemInit(MemoryPool, sizeof(MemoryPool), 16))
  {
      printf("Failure during memory pool initialization.\n");
     return 0;
  }
```

Notes

```
    /* Allocate single block */
    Block = stFixedMemAlloc(MemoryPool);
    if(!Block)
    {
        printf("Failure during memory block allocation.\n");
      return 0;
    }

    /* ... */

    /* Free memory block */
    stFixedMemFree(MemoryPool, Block);

    /* ... */

    /* Deinitialization */
    arDeinit();

    return 0;
}
```

5.3.1 Fixed Size Memory Management Configuration

ST_USE_FIXMEM definition

The ST_USE_FIXMEM constant comprises information whether functions for fixed size memory blocks allocation are incorporated or expelled from compilation. Functions will be compiled if the constant value is fixed for 1 and will not be compiled if the constant value is fixed for 0.

When this constant value is not defined, the value will be 1, by default. It will cause the addition of functions into output code. If the fixed size memory management functions are not utilized, it can be expelled from compilation by setting ST_USE_FIXMEM for 0 to decrease output code.

Functions

stFixedMemAlloc function

Declaration:

```
PVOID stFixedMemAlloc(
   PVOID MemoryPool
);
```

Parameters:

MemoryPool

Notes

It defines the address of the memory pool.

Return value:

It indicates the pointer to newly assigned memory block or **NULL** on failure.

Description:

Function assigns a memory block in stated memory pool. Block size is stated during memory pool initialization. Allocated addresses are always allocated to value stated in **AR_MEMORY_ALIGNMENT**, comparatively to start of the memory pool. If function successes, it return pointer to newly assigned memory block. On failure, it returns a **NULL**. Function do not sets the previous error code on failure. Access to the fixed memory is never synchronized. If it is compulsory it should be executed.

Function will be expelled from compilation when functions accountable for fixed size memory management are disabled, by setting **ST_USE_FIXMEM** for 0.

stFixedMemFree function

Declaration:

```
BOOL  stFixedMemFree(
   PVOID  MemoryPool,
   PVOID  Address
);
```

Parameters:

MemoryPool

It defines the address of memory pool.

Address

It indicates the address of the block to release.

Return value:

TRUE on success or **FALSE** on failure.

Description:

Function releases memory block assigned by **stFixedMemAlloc** function. It does not set the last error code on failure. Access to the fixed memory is never coordinated. If it is compulsory that it should be implemented.

Function will be expelled from compilation when functions accountable for memory management are disabled, by setting **ST_USE_FIXMEM** for 0.

stFixedMemInit function

Declaration:

```
BOOL  stFixedMemInit(
   PVOID  MemoryPool,
   SIZE  MemorySize,
```

Notes

```
    SIZE BlockSize
);
```

Parameters:

MemoryPool

It defines the address of memory pool to be initialized.

MemorySize

It indicates the total size of the memory.

BlockSize

It indicates the size of the memory block that will be allocated.

Return value:

TRUE on success or **FALSE** on failure.

Description:

Function initializes memory pool, for particular memory address. The memory address must be passed to stFixedMemAlloc and stFixedMemFree functions to assign memory blocks in this memory. Size of the every allocated block is at all times fixed. This function does not set the last error code on failure. Access to the fixed memory is never coordinated. If it is mandatory it should be implemented.

Function will be not compiled when functions accountable for memory management are disabled, by setting **ST_USE_FIXMEM** for 0.

Task Make distinction between *stFixedMemFree function* and *stFixedMemInit function*.

Self Assessment

Fill in the blanks:

6. ..... blocks allocation algorithm has been established to permit technique of memory allocation in static time, separately to number of allocated blocks.
7. The ..... constant comprises information whether functions for fixed size memory blocks allocation are incorporated or expelled from compilation.

5.4 Changing the Size of a Memory Block

Frequently you do not know for certain how large a block you will eventually need at the time you must start to use the block.

Example: The block might be a buffer that you use to hold a line being read from a file; regardless of how long you make the buffer originally, you may encounter a line that is longer.

You can make the block longer by calling realloc. This function is affirmed in 'stdlib.h'.

Function: void * **realloc** *(void *ptr, size_t newsize)*

The realloc function modifies the size of the block whose address is *ptr* to be *newsize.*

Notes

As the space after the end of the block may be in use, realloc may find it essential to copy the block to a new address where more free space is obtainable. The value of realloc is the new address of the block. If the block is required to be moved, realloc copies the old contents.

If you pass a null pointer for *ptr*, realloc behaves just like 'malloc (*newsize*)'. This can be expedient, but be cautious that older implementations (before ANSI C) may not sustain this behavior, and will perhaps crash when realloc is passed a null pointer.

Similar to malloc, realloc may return a null pointer if no memory space is obtainable to make the block bigger. When this occurs the original block is untouched; it has not been customized or relocated.

In many cases it makes no distinction what happens to the original block when realloc fails, since the application program cannot continue when it is out of memory, and the only thing to do is to provide a fatal error message. Over and over again it is suitable to write and use a subroutine, conventionally known as xrealloc, that takes care of the error message as xmalloc does for malloc:

```
void *
xrealloc (void *ptr, size_t size)
{
   register void *value = realloc (ptr, size);
   if (value == 0)
      fatal ("Virtual memory exhausted");
   return value;
}
```

You can also utilize realloc to make a block smaller. The reason you would do this is to evade tying up a lot of memory space when only a little is required. Making a block smaller at times necessitates copying it, so it can fail if no other space is obtainable.

Did u know? If the new size you state is the similar as the old size, realloc is assured to modify nothing and return the similar address that you gave.

Self Assessment

Fill in the blanks:

8. The ..... function modifies the size of the block whose address is *ptr* to be *newsize*.
9. If you pass a null pointer for *ptr*, realloc behaves just like '..... '.
10. Similar to malloc, realloc may return a ..... pointer if no memory space is obtainable to make the block bigger.

5.5 Using LocalReAlloc()

The LocalReAlloc function modifies the size or the attributes of a stated local memory object. The size can increase or decrease.

HLOCAL LocalReAlloc(

```
     HLOCAL hMem,     // handle of local memory object
```

Notes

```
    UINT uBytes,     // new size of block
    UINT uFlags      // how to reallocate object
  );
```

5.5.1 Parameters

hMem

It determines the local memory object to be reallocated. This handle is returned by either the LocalAlloc or LocalReAlloc function.

UBytes

It states the new size, in bytes, of the memory block. If this parameter is zero and the uFlags parameter states the LMEM_MOVEABLE flag, the function returns a handle to a memory object that is marked as discarded. If uFlagsspecifies the LMEM_MODIFY flag, this parameter is ignored.

uFlags

It specifies how to reallocate the local memory object. If the LMEM_MODIFY flag is specified, this parameter modifies the attributes of the memory object, and the uBytes parameter is unobserved. Or else, this parameter handles the reallocation of the memory object.

The LMEM_MODIFY flag can be united with either or both of the following flags:

Flag	Meaning
LMEM_DISCARDABLE	Assigns discardable memory if the LMEM_MODIFY flag is also specified. This flag is overlooked, unless the object was previously allocated as movable or the LMEM_MOVEABLE flag is also specified.
LMEM_MOVEABLE	You cannot merge LMEM_MOVEABLE with LMEM_MODIFY to change a fixed memory object into a movable one. The function returns an error if an application attempts this.

If uFlags does not state LMEM_MODIFY, this parameter can be any mixture of the following flags:

Flag	Meaning
LMEM_MOVEABLE	If uBytes is zero, discards a formerly movable and discardable memory block. If the objects lock count is not zero or the block is not movable and discardable, the function fails.
	If uBytes is nonzero, enables the system to move the reallocated block to a new location without modifying the movable or fixed attribute of the memory object. If the object is fixed, the handle returned may be dissimilar from the handle stated in the hMem parameter. If the object is movable, the block can be moved without invalidating the handle, even if the object is currently locked by a previous call to the LocalLockfunction. To obtain the new address of the memory block, use LocalLock.
LMEM_NOCOMPACT	Avoids memory from being compacted or discarded to satisfy the allocation request.
LMEM_ZEROINIT	Causes the supplementary memory contents to be initialized to zero if the memory object is growing in size.

Notes

Task Make distinction between *uBytes and uFlags.*

5.5.2 Return Values

If the function succeeds, the return value is the handle of the reallocated memory object.

If the function fails, the return value is NULL. To obtain extended error information, call GetLastError.

Notes If LocalReAlloc reallocates a movable object, the return value is the handle of the memory object. To translate the handle to a pointer, make use of the LocalLock function. If LocalReAlloc reallocates a fixed object, the value of the handle returned is the address of the first byte of the memory block. To access the memory, a process can just cast the return value to a pointer.

Self Assessment

Fill in the blanks:

11. The ..... function modifies the size or the attributes of a stated local memory object.
12. ..... determines the local memory object to be reallocated.
13. ..... specifies how to reallocate the local memory object.

5.6 Discardable Memory Blocks

Windows can reallocate a discardable memory block to a 0 length when it requires space to gratify another allotment request. Performing this destroys all data enclosed in the memory block. You usually use a discardable memory block to hold information that is suitable to keep in memory but that can simply be recreated when required.

You cannot obtain a discardable memory block by means of malloc or HeapAlloc.

Caution You must either use GlobalAlloc with the GMEM_MOVEABLE and GMEM_DISCARDABLE flags.

When you assign a discardable memory block, Windows returns a handle to the block. When you lock the block, Windows returns the address of the block. The handle returned when allocating a discardable memory block remains suitable even after the block is discarded; this can happen at any time the block is unlocked.

The function call in this case turns out to be

```
HANDLE  hMem;

hMem=GlobalAlloc(GMEM_MOVEABLE|GMEM_DISCARDABLE,10);
```

Notes

The function call for GlobalLock remains as in portable memory blocks.

A discardable resource can be substituted by reloading the resource form the resource segment of the module. This occurs transparently. Specifically, if you call a resource loading function and the obtainable copy of the resource has been discarded, a new copy will be loaded.

Self Assessment

Fill in the blanks:

14. Windows can reallocate a ..... memory block to a 0 length when it requires space to gratify another allotment request.
15. A discardable resource can be substituted by ..... the resource form the resource segment of the module.

5.7 Summary

- Global memory exists in device memory and device memory is used through 32-, 64-, or 128-byte memory transactions.
- Global memory instructions assist reading or writing words of size equal to 1, 2, 4, 8, or 16 bytes.
- Local memory accesses only happen for some automatic variables.
- On devices of compute capability 2.0, local memory accesses are at all times cached in L1 and L2 in the similar manner as global memory accesses
- Fixed size memory blocks allocation algorithm has been established to permit technique of memory allocation in static time, separately to number of allocated blocks.
- All of the allocated blocks have a fixed size, defined throughout memory initialization.
- The ST_USE_FIXMEM constant comprises information whether functions for fixed size memory blocks allocation are incorporated or expelled from compilation.
- The realloc function modifies the size of the block whose address is *ptr* to be *newsize*.
- The LocalReAlloc function modifies the size or the attributes of a stated local memory object.
- Windows can reallocate a discardable memory block to a 0 length when it requires space to gratify another allotment request.

5.8 Keywords

Discardable memory block: Windows can reallocate a discardable memory block to a 0 length when it requires space to gratify another allotment request.

LocalReAlloc: The LocalReAlloc function modifies the size or the attributes of a stated local memory object.

realloc: The realloc function modifies the size of the block whose address is *ptr* to be *newsize*.

ST_USE_FIXMEM: The ST_USE_FIXMEM constant comprises information whether functions for fixed size memory blocks allocation are incorporated or expelled from compilation.

Notes

5.9 Review Questions

1. Make distinction between local memory and global memory.
2. Illustrate the Size and Alignment Requirement in global memory.
3. What are local memory bocks? Illustrate their functions.
4. What are the steps used in assigning and using memory?
5. Explain the concept of using Fixed Memory Blocks with examples.
6. What is stFixedMemAlloc function? Define its parameters and return values.
7. Illustrate how to declare the function stFixedMemInit function. Also discuss its parameters and return values.
8. Explain the function used for changing the size of a memory block.
9. What are the various parameters used in LocalReAlloc()? Illustrate.
10. Windows can reallocate a discardable memory block to a 0 length when it requires space to gratify another allotment request. Comment.

Answers: Self Assessment

1.	Global	2.	specifiers
3.	Local	4.	heap
5.	LocalAlloc	6.	Fixed size memory
7.	ST_USE_FIXMEM	8.	realloc
9.	malloc (newsize)	10.	null
11.	LocalReAlloc	12.	*hMem*
13.	*uFlags*	14.	discardable
15.	reloading		

5.10 Further Readings

Books

Brent E. Rector, *Win32 Programming,* Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

ldp.org/LDP/tlk/mm/memory.html

Notes

Unit 6: Memory Management (II)

CONTENTS

Objectives

Introduction

6.1 Global Memory Allocation

6.2 What Windows is actually Doing with Memory?

6.3 System Memory and System Resources

6.3.1 What are System Resources?

6.3.2 Resource Comparison

6.4 Summary

6.5 Keywords

6.6 Review Questions

6.7 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of global memory allocation
- Discuss what windows is actually doing with memory
- Understand system memory and system resources

Introduction

Memory allocation and de-allocation takes place at several times in database. Here you will recognize the process of assigning global memory. The term System Resources basically includes two major areas of Windows memory that are reserved for and used by particular Windows components. You will also discuss in this unit the concept of system memory and system resource.

6.1 Global Memory Allocation

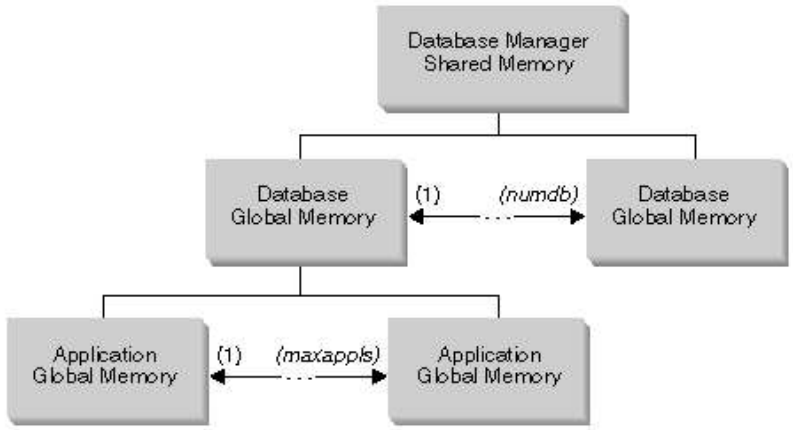
Memory may be assigned to a specific memory area when a stated event happens, like when an application connects, or it may be reallocated depending on a alteration in a configuration parameter setting.

The Figure 6.1 displays the dissimilar areas of memory that the database manager assigns for numerous uses and the configuration parameters that permit you to manage the size of this memory.

Notes In an Enterprise Server Edition environment that includes numerous logical database partitions, every database partition contains its own Database Manager Shared Memory set.

Notes

Figure 6.1: Types of Memory Utilized by the Database Manager



Memory is assigned for every instance of the database manager when the following events take place:

- ***When the database manager is started:*** Database manager global shared memory is assigned and remains assigned until the database manager is stopped. This area includes information that the database manager utilizes to handle activity across all database connections. Instance shared memory can be handled by the *instance_memory* configuration parameter. By default, this parameter is fixed to automatic in order that DB2 computes the amount of memory assigned for the instance.
- ***When a database is activated or linked to for the first time:*** Database global memory is assigned. Database global memory is utilized across all applications that attach to the database. The size of the database global memory is stated by the *database_memory* configuration parameter. By default, this parameter is fixed to *automatic* in order that DB2 computes the quantity of memory assigned for the database. You can fix *database_memory* to assign more memory than is required originally so that the additional memory can be dynamically distributed afterwards.
  - Even though the total quantity of database global memory cannot be increased or decreased while the database is active, memory for areas enclosed in database global memory can be accustomed.

Example: The following memory areas can be dynamically accustomed to decrease memory allocated to one area and augment memory in another area.

- Buffer pools (using the ALTER BUFFERPOOL DDL statement)
- Database heap (including log buffers)
- Utility heap
- Package cache
- Catalog cache
- Lock list

In the surroundings in which the database manager intra-partition parallelism configuration parameter (intra_parallel) is enabled, or in the surroundings in which the connection concentrator is enabled, the shared sort heap is also assigned as piece of the database global memory.

Notes

- ***When an application connects to a database:*** In a partitioned database surroundings, in a non-partitioned database with the database manager intra-partition parallelism configuration parameter (*intra_parallel*) enabled, or in the surroundings in which the connection concentrator is enabled, multiple applications can be assigned to *application groups* to share memory. Every application group has its own allotment of shared memory. In the application-group shared memory, each application contains its own application control heap but utilizes the shared heap of the application group which enhances the competence of cache and memory usage.

  The following three database configuration parameters find out the size of the application group memory:

  - The *appgroup_mem_sz* parameter, which states the size of the shared memory for the application group
  - The *groupheap_ratio* parameter, which states the percent of the application-group shared memory permitted for the shared heap
  - The *app_ctl_heap_sz* parameter, which states the size of the control heap for every application in the group.

  The database manager configuration parameter *max_connections* fixes an upper limit on the number of applications that can attach to a database. As each application that attaches to a database includes the allocation of some memory, permitting a larger number of concurrent applications will potentially make use of more memory.

  To some extent, the maximum number of applications is also administered by the database manager configuration parameter *maxagents* or *max_coordagents* for partitioned database surroundings. The *maxagents* parameter fixes an upper limit to the total number of database manager agents in a database partition. These database manager agents comprise active coordinator agents, subagents, inactive agents, and idle agents.

  If you are encountering memory errors when attempting to connect to a database, try making the following configuration amendments:

  - In a non-partitioned database environment with no intra-query parallelism enabled, enlarge the value of the maxagents database configuration parameter.
  - In a partitioned database environment or an environment where intra-query parallelism is enabled, augment the larger of maxagents or max_coordagents.
  - In surroundings where max_connections has been configured to a value that is greater than max_coordagents, you can also augment max_connections to resolve the error.

- ***When an agent is created:*** Agent private memory is assigned for an agent when the agent is allocated as the consequence of a connect request or a new SQL request in a parallel environment. Agent private memory is assigned for the agent and comprises memory that is used only by this particular agent, like the sort heap and the application heap.

  Caution When a database is already in use by one application, only agent private memory and application global shared memory is assigned for consequent connecting applications.

The figure also states the following configuration parameter settings, which limit the amount of memory that is assigned for every type of memory area. Observe that in a partitioned database environment, this memory is assigned on each database partition.

Notes

- ***instance_memory:*** This parameter states how much memory is assigned for instance management.
- ***numdb:*** This parameter states the maximum number of simultaneous active databases that dissimilar applications can use. Since each database has its own global memory area, the quantity of memory that might be assigned increases if you increase the value of this parameter.
- ***maxappls:*** This parameter states the maximum number of applications that can concurrently connect to a single database. It affects the amount of memory that might be assigned for agent private memory and application global memory for that database.

Did u know? This parameter can be set in a different way for every database.

- ***maxagents and max_coordagents for parallel processing:*** These parameters limit the number of database manager agents that can subsist concurrently across all active databases in an instance. Together with *maxappls*, these parameters limit the amount of memory assigned for agent private memory and application global memory.

The memory tracker, invoked by the db2mtrk command, permits you to observe the current assignment of memory inside the instance, involving the following types of information for every memory pool:

- Current size
- Maximum size (hard limit)
- Largest size (high water mark)

On Unix and Linux, even though the ipcs command can be utilized to list all the shared memory segments, it does not precisely reflect the amount of resources taken. You can use the db2mtrk command as an substitute to ipcs.

Task Make distinction between Numdb and maxappls parameters.

Self Assessment

Fill in the blanks:

1. Memory may be assigned to a specific memory area when a stated .....happens.
2. In an Enterprise Server Edition environment that includes numerous logical database partitions, every database ..... contains its own Database Manager Shared Memory set.
3. The ..... parameter states the size of the shared memory for the application group
4. An ..... parameter states how much memory is assigned for instance management.
5. .....parameter states the maximum number of applications that can concurrently connect to a single database.
6. The database manager configuration parameter ..... fixes an upper limit on the number of applications that can attach to a database.

Notes

7. The size of the database global memory is stated by the ..... configuration parameter.

8. You can fix ..... to assign more memory than is required originally so that the additional memory can be dynamically distributed afterwards.

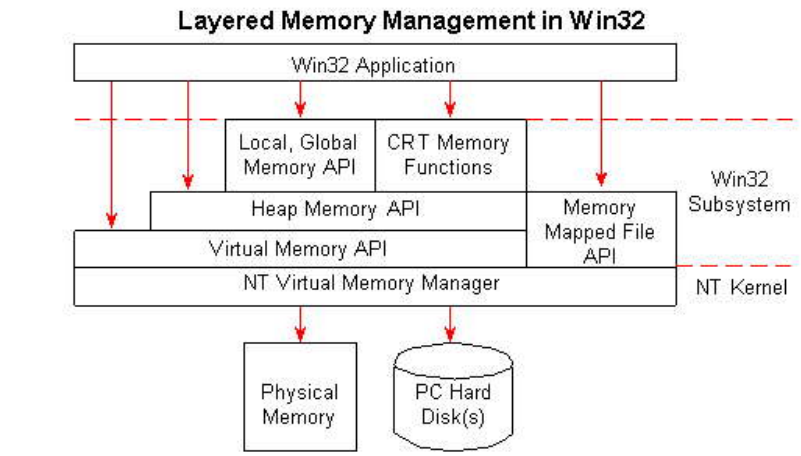
9. The..... parameter fixes an upper limit to the total number of database manager agents in a database partition.

6.2 What Windows is actually Doing with Memory?

The first version of the Windows operating system established a technique of managing dynamic memory relying on a single *global heap*, which all applications and the system share, and several, private *local heaps*, one for every application. Local and global memory management functions were also offered, providing extended traits for this new memory management system. More lately, the Microsoft C run-time (CRT) libraries were customized to comprise capabilities for administrating these heaps in Windows by means of native CRT functions like **malloc** and **free**. Accordingly, developers are now left with an option—study the new application programming interface (API) offered as part of Windows or stick to the transportable, and usually familiar, CRT functions for managing memory in applications written for Windows.

Did u know? Window provides three groups of functions for managing memory in applications: memory-mapped file functions, heap memory functions, and virtual-memory functions.

Figure 6.2: The Windows API offers different Levels of Memory Management for Adaptability in Application Programming



Altogether, six sets of memory management functions subsist in Windows, as displayed in Figure 6.2, all of which were intended to be used separately of one another. So which set of functions should you utilize The answer to this question relies deeply on two things: the type of memory management you desire and how the functions pertinent to it are executed in the operating system. Alternatively, are you constructing a large database application where you plan to influence subsets of a large memory structure? Or maybe you're arranging some simple dynamic memory structures, like linked lists or binary trees? In both cases, you are required to

know which functions suggest the traits best matched to your intention and precisely how much of a resource hit appears when using every function.

Self Assessment

Fill in the blanks:

10. The first version of the Windows operating system established a technique of managing dynamic memory relying on a single ..... *heap*, which all applications and the system share, and several, private *local heaps*, one for every application.
11. The Windows API offers different levels of ..... management for adaptability in application programming.

6.3 System Memory and System Resources

Consider that you've got 4GHz Pentium IV with 1Gb of RAM and Windows 98. Still, applications appear to be running out of memory. They run sluggishly, or you may obtain the dreaded "Unable to create control" message, or poorer yet the whole operating system freezes up. What is basically the problem?

The short respond resources. You've got **gigabyte** of RAM! The short respond: that makes no dissimilarity. You add another 512Mb of RAM. Now you contain 1.5Gb of RAM. Still you have the similar problem. The depressing truth is, the amount of physical memory has completely no impact on system resources. When your system runs short on system resources awful things, such as the foregoing, occur.

Notes While it appears to be picking on Windows98 here, the memory model is fundamentally the similar in Windows95, Windows98 and Windows Me. Afterwards, we'll talk regarding WindowsNT and its successors Windows 2000 and WindowsXP, which share a considerably dissimilar memory model.

6.3.1 What are System Resources?

The term System Resources essentially comprises two major areas of Windows memory that are reserved for and used by particular Windows components. They are known as User and GDI. User resources points to the input manager user32.dll. It manages input from your mouse, keyboard, and other sources, like communication ports, file handles, etc. GDI symbolizes Graphics Device Interface and is in accuse of the noticeable components of Windows. It accumulates fonts, brushes, bitmaps, and other graphics stuff, in addition to lends support to other graphic output devices like printers. On Windows 9x/Me systems, you can observe resources by means of the resource monitor. It returns the amount of free resources as a percentage.

Under Windows 3.x, these two areas of memory were restricted to 64K. All running applications shared that 64K for User resources and 64K for GDI. Unnecessary to say, that formed a huge bottleneck.

With the foreword of Windows95 and enduring through Windows98 and WindowsMe, that 64K was augmented dramatically and each of the two areas was further subdivided as follows:

- The 16-bit User heap (64K).

Notes

- The 32-bit User window heap (2MB).
- The 32-bit User menu heap (2MB).
- The 16-bit GDI heap (64K).
- The 32-bit GDI heap (2MB).

However, each one of these five memory segments is still fixed in size because of the Windows 9x/Me architecture, and cannot be increased, irrespective of the amount of physical memory (RAM) installed on the machine. Modern applications stipulate more and more of the system. The more controls an application generates and the more files it opens, the more stress is positioned on the operating system.

Now, we have this difficulty regarding resources, but wait; it gets worse. When you first install the operating system and start the computer, you perhaps have somewhere in the neighborhood of 96%-98% free system resources. Over time, you install applications and utilities. Now, when you first start the computer you may only have 70%-80% free resources. This is because many applications install minute programs that are started when the system starts. Each of these utilizes resources.

As you run a variety of applications, each application further reduces these system resources. Hypothetically, when an application concludes, the resources it used should be returned to the operating system to utilize for other applications. In the actual world, this doesn't always take place. In some cases, it is normal. Some shared resources are not loaded until an application requests them. Those are not usually released when the application terminates. They are kept loaded in memory to permit the next application quicker access to them.

Some applications, though, do not behave properly. They may not free all the resources they assign. This is known as resource leakage. Here, a block of memory is marked by the operating system as being in use and it cannot be utilized by the operating system, or any other application. When this takes place, the only means to recover that area of memory is to reboot the computer.

The fact is, if you are a home user who runs a word processor, a spreadsheet, an Internet browser and an e-mail client program, the solution is that it is not that huge problem. Those programs are usually not horribly resource intensive and, if you reboot the computer frequently, you'll probably never have a trouble. If you obtain the dreaded "Unable to create control" error, rebooting the machine will usually free resources that have been lost by misbehaving applications and permit your computer to function usually.

Just keep in mind that applications are sharing memory here. If you encounter a system error, such as a General Protection Fault, there is a high probability that system memory has been tainted. You should straight away put aside any open documents and reboot the computer if you want the system to remain stable. The computer may *emerge* to function generally after such an error, but it is undependable and may cause an apparently unconnected error later on.

Caution If you obtain a system error, you should reboot the computer.

If, alternatively, you want to run some more resource rigorous applications, the Windows 95/98/Me memory model will be a steady source of problems for you.

Example: A high-end graphics editor, or a program or web development environment, etc.

If you have a comparatively new, clean install of the operating system and you haven't installed a lot of other programs and you don't have a lot of fonts installed and you don't run too many

Notes

applications simultaneously, etc., you may get away with it. Sooner or later, though, it will become a problem.

Now we've defined the problem. What is the solution? If you build up software, web sites , or graphics (by means of high end software), the answer, even though you may not like it, is to advance the operating system. The WindowsNT memory model is the answer. WindowsNT, Windows2000 and WindowsXP share a hugely different memory model. Under this NT based memory management system, resources are not restricted. As long as you have sufficient physical memory (RAM) or virtual memory (disk space) your applications will not run out of resources. The WindowsNT memory system assigns resources dynamically. As long as the memory is obtainable, your application can have it.

Even better, every application obtains its own virtual copy of the operating system. That means that, applications are isolated and, if an error appears, system memory is not corrupted. When an application terminates, all the memory assigned to it is released back to the operating system. That means that resource, or memory leakage is also almost nonexistent. It is likely for an application to incessantly demand more and more memory and the system will provide it, until memory turns out to be short, but closing the application releases all that memory without the requirement to reboot the machine.

The following chart displays the memory assigned for each particular purpose by the various operating systems:

6.3.2 Resource Comparison

Window/Menu Handles	about 200	32KB (each)	Unlimited
Timers	32	Unlimited	Unlimited
COM/LPT ports	4 each	Unlimited	Unlimited
Listbox items (per listbox)	8KB	32KB	Unlimited
Listbox data (per listbox)	64KB	Unlimited	Unlimited
Edit control data (per control)	64KB	Unlimited	Unlimited
Regions	All in 64KB segment	Unlimited	Unlimited
Logical pens, brushes	All in 64KB segment	64KB segment	Unlimited
Physical pens, brushes	All in 64KB segment	Unlimited	Unlimited
Logical fonts	All in 64KB segment	750-800	Unlimited
Installed fonts	250-300 (best case)	1000	Unlimited
Device Contexts	200 (best case)	16KB	Unlimited

While Windows95, Windows98 and WindowsMe operating systems enhanced memory management enormously over Windows 3.1, they were still intended for the home user. They were never proposed to be a severe development platform. Coming across errors is a part of software development. An operating system that is not fault tolerant has no place in the software development environment. If you develop software, or web sites by means of a serious development tool, you should be using WindowsNT, Windows2000, or WindowsXP.

Task Illustrate the function of User resources.

Notes

Self Assessment

Fill in the blanks:

12. The term .....essentially comprises two major areas of Windows memory that are reserved for and used by particular Windows components.
13. ..... resources points to the input manager user32.dll.
14. .....symbolizes Graphics Device Interface and is in accuse of the noticeable components of Windows.
15. An operating system that is not ..... tolerant has no place in the software development environment.

6.4 Summary

- Memory may be assigned to a specific memory area when a stated event happens, like when an application connects, or it may be re-allocated depending on a alteration in a configuration parameter setting.
- When the database manager is started, database manager global shared memory is assigned and remains assigned until the database manager is stopped.
- In the surroundings in which the database manager intra-partition parallelism configuration parameter (intra_parallel) is enabled, or in the surroundings in which the connection concentrator is enabled, the shared sort heap is also assigned as piece of the database global memory.
- *Numdb* parameter states the maximum number of simultaneous active databases that dissimilar applications can use.
- The memory tracker, invoked by the db2mtrk command, permits you to observe the current assignment of memory inside the instance.
- The first version of the Windows operating system established a technique of managing dynamic memory relying on a single *global heap*, which all applications and the system share, and several, private *local heaps*, one for every application.
- The term System Resources essentially comprises two major areas of Windows memory that are reserved for and used by particular Windows components. They are known as User and GDI.
- As you run a variety of applications, each application further reduces these system resources.

6.5 Keywords

Maxappl: This parameter states the maximum number of applications that can concurrently connect to a single database.

Numdb: This parameter states the maximum number of simultaneous active databases that dissimilar applications can use.

System Resources: The term System Resources essentially comprises two major areas of Windows memory that are reserved for and used by particular Windows components.

Notes

6.6 Review Questions

1. Elucidate the concept of global memory allocation.
2. Illustrate various events that take place when memory is assigned for every instance of the database manager.
3. What are the different database configuration parameters that find out the size of the application group memory?
4. What configurations adjustments are to be made if encountering memory errors while attempting to connect to a database? Discuss.
5. Explain the concept of managing memory in Windows operating system.
6. Illustrate the concept of system resources and system memory.
7. What are the two areas of memory that are restricted to 64K? Illustrate.
8. Generate a chart that displays the memory assigned for each particular purpose by the various operating systems.
9. Make distinction between user and GDI.
10. As you run a variety of applications, each application further reduces these system resources. Comment.

Answers: Self Assessment

1.	event	2.	partition
3.	appgroup_mem_sz	4.	instance_memory
5.	Maxappls	6.	max_connections
7.	database_memory	8.	database_memory
9.	maxagents	10.	global
11.	memory	12.	System Resources
13.	User	14.	GDI
15.	fault		

6.7 Further Readings

Books

Brent E. Rector, *Win32 Programming,* Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

ldp.org/LDP/tlk/mm/memory.html

Notes

Unit 7: Character Sets, Fonts and the Keyboard

CONTENTS

Objectives
Introduction
7.1 The ANSI Character Set
7.2 Trying the Character Functions
7.2.1 Case Conversion Functions
7.2.2 Character Manipulation Functions
7.3 Keyboard Message Processing
7.4 WM_CHAR Message
7.4.1 Parameters
7.4.2 Return Value
7.5 System Key Messages and Dead Characters
7.6 Implementing a Simple Keyboard Interface
7.7 Selecting a Stock Font
7.8 Using Logical Fonts
7.9 Text Metric
7.9.1 Members
7.10 Putting Fonts to Work
7.10.1 Installing OpenType or TrueType Fonts in Windows
7.10.2 Installing PostScript Type 1 Fonts in WindowsXP or 2000
7.11 Keyboard Accelerators
7.11.1 Accelerator Tables
7.11.2 Accelerator Table Creation
7.11.3 Accelerator Keystroke Assignments
7.11.4 Accelerators and Menus
7.11.5 UI State
7.12 Summary
7.13 Keywords
7.14 Review Questions
7.15 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of the ANSI Character Font and trying character functions
- Discuss keyboard message processing
- Illustrate WM_CHAR message

Notes

- Discuss System key messages and Dead Characters
- Understand implementing a simple keyboard interface
- Discuss stock font, logical fonts, text metric, and putting fonts to work
- Understand keyboard accelerators

Introduction

A character set is a conventionalized relationship among letters and character positions. Most fonts utilize one of numerous standard character sets, but there are also fonts (generally symbol fonts) with random mappings. A font is a set of graphic elements known as *glyphs* (letters or symbols) mapped onto a set of character positions (numbers). Keyboard layout is a conventionalized relationship among keyboard keys and character positions. In this unit, we will discuss various concepts related to Character Sets, Fonts, and the Keyboard.

7.1 The ANSI Character Set

ANSI symbolizes American National Standards Institute. The ANSI character set involves the standard ASCII character set (values 0 to 127), in addition to extended character set (values 128 to 255).

ANSI character set is defined list of characters identified by the computer hardware and software. Every character is represented by a number.

Example: The ASCII character set utilizes the numbers 0 through 127 to symbolize all English characters in addition to special control characters.

European ISO character sets are alike to ASCII, but they enclose supplementary characters for European languages.

The ANSI character set defines 224 characters (32 to 255 Decimal, 20 to FF Hexadecimal). Characters 32 to 127 are shared with the ASCII Character Set and characters 128 to 255 are shared with the ISO Latin-1 character set utilized by Web browsers.

ANSI Character Set

HEX		00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
HEX	**DEC**	**0**	**16**	**32**	**48**	**64**	**80**	**96**	**112**	**128**	**144**	**160**	**176**	**192**	**208**	**224**	**240**
0	**0**			space	0	@	P	`	p	€	ê	nbsp	°	À	Ð	à	ð
1	**1**			!	1	A	Q	a	q	Å	‘	¡	±	Á	Ñ	á	ñ
2	**2**			"	2	B	R	b	r	‚	’	¢	²	Â	Ò	â	ò
3	**3**			#	3	C	S	c	s	ƒ	“	£	³	Ã	Ó	ã	ó
4	**4**			$	4	D	T	d	t	„	”	¤	´	Ä	Ô	ä	ô
5	**5**			%	5	E	U	e	u	…	•	¥	µ	Å	Õ	å	õ
6	**6**			&	6	F	V	f	v	†	–	¦	¶	Æ	Ö	æ	ö
7	**7**			'	7	G	W	g	w	‡	—	§	·	Ç	×	ç	÷
8	**8**			(	8	H	X	h	x	ˆ	˜	¨	¸	È	Ø	è	ø
9	**9**	TAB		)	9	I	Y	i	y	‰	™	©	¹	É	Ù	é	ù
A	**10**	LF		*	:	J	Z	j	z	Š	š	ª	º	Ê	Ú	ê	ú
B	**11**			+	;	K	[	k	{	‹	›	«	»	Ë	Û	ë	û
C	**12**			,	<	L	\	l	|	Œ	œ	¬	¼	Ì	Ü	ì	ü
D	**13**	CR		-	=	M	]	m	}	ç	ù		½	Í	Ý	í	ý
E	**14**			.	>	N	^	n	~	Ž	ž	®	¾	Î	Þ	î	þ
F	**15**			/	?	O	_	o	□	è	Ÿ	¯	¿	Ï	ß	ï	ÿ

Notes

Notes On many systems the characters 127 to 160 are regarded as control characters and that these do not map straightforwardly to a UNICODE (UTF-8) character as does the remaining characters.

Self Assessment

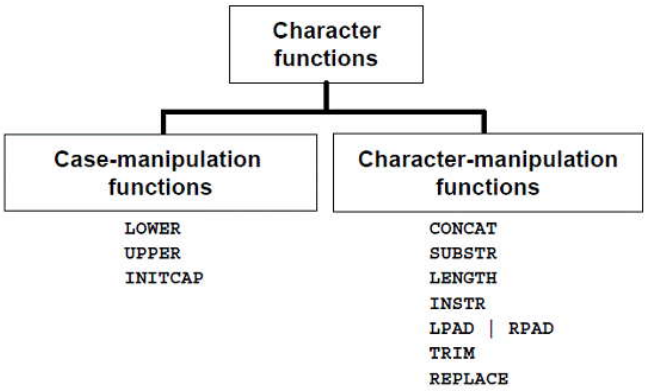
Fill in the blank:

1. ..... is defined list of characters identified by the computer hardware and software.

7.2 Trying the Character Functions

Character functions function on values of character class datatype such as Char, Varchar2, Varchar, etc. These functions can return either character class datatype or number class datatype depending on the operation accomplished on the input data. Length of a value returned by these functions is restricted up to 4000 bytes for varchar2 datatype and 2000 bytes for char datatype. If a function returns value that surpasses the length limit, Oracle routinely trim the value before returning the consequence. Some of the SQL incorporated character functions are specified in the following table.

Figure 7.1: Single-row Character Functions



CHR	**CONCAT**	**INITCAP**	**SUBSTR**
RTRIM	**LTRIM**	**TRIM**	**REPLACE**
LPAD	**RPAD**	**UPPER**	**LOWER**
TRANSLATE	**ASCII**	**INSTR**	**LENGTH**

7.2.1 Case Conversion Functions

The three case conversion functions are LOWER, UPPER, and INITCAP.

- ***Lower:*** It converts mixed case or uppercase character string lo lowercase
- ***Upper:*** It converts mixed case or lowercase character string to uppercase
- ***Initcap:*** It converts first letter of each word to uppercase and other letters to lowercase.

Notes

Example: Show the employee number, name, and department number for employee John

```
SELECT   empno, ename, deptno
FROM     emp
WHERE    ename = 'john';
no rows selected
```

The WHERE clause of the first SQL statement specifies the employee name as 'john'. As all the data in the EMP table is amassed in uppercase. The name 'john' does not locate a match in the EMP table and as a consequence no rows are chosen.

```
SELECT   empno, ename, deptno
FROM     emp
WHERE    LOWER( ename) = 'blake' ;
```

EMPNO	ENAME	DEPTNO
7698	JOHN	30

The WHERE clause of the second SQL statement specifies that the employee name in the EMP table be converted to lowercase and then be compared to 'john'. Since both the names are in lowercase now, a match is located and one row is chosen.

Task Make distinction between LOWER function and UPPER function.

7.2.2 Character Manipulation Functions

Function	Result
CONCAT('Good' , 'String')	GoodString
SUBSTR('String' , 1,3)	Str
LENGTH('String')	6
INSTR('String', 'r')	3
LPAD(sal,10 , '*')	******5000

CONCAT. SUBSTR, LENGTH, INSTR, and LPAD are the five character manipulation functions discussed here.

Concat: It unite values together (You are restricted to using two parameters with CONCAT.)

Substr: It takes out a string of determined length

Length: It exhibits the length of a string as a numeric value

Instr: It locates numeric position of a named character

Lpad: It Pads the character value right-justified

Did u know? RPAD character manipulation function pads the character value left-justified.

Notes

Example: Ernployee name and job joined together, length of the employee name, and the numeric position of the letter A in the employee name, for all employees who are in sales.

```
SELECT ename, CONCAT( ename, job),
LENGTH(ename),
INSTR(ename, 'A')
FROM emp
WHERE SUBSTR(job, 1, 5) = 'SALES' ;
```

Ename	Concat(ename,job)	Length(ename)	Instr(ename,'a')
Allen	Allensalesman	5	1
Ward	Wardsalesman	4	2
Martin	Martinsalesman	6	2
Turner	Turnersalesman	6	0

Self Assessment

Fill in the blanks:

2. ..... function converts mixed case or uppercase character string lo lowercase.
3. ..... function locates numeric position of a named character.

7.3 Keyboard Message Processing

A window obtains keyboard input as keystroke messages and character messages. Keystroke messages handle window behavior and character messages verify the text that is exhibited in a window.

Windows Embedded CE produces a WM_KEYDOWN or a WM_SYSKEYDOWN message when the user presses a key. If the user holds a key down long enough to begin automatic repeat functionality, the system produces repeated WM_KEYDOWN or WM_SYSKEYDOWN messages. When the user releases a key, the system produces a WM_KEYUP or a WM_SYSKEYUP message.

The system creates a dissimilarity between system keystrokes and nonsystem keystrokes. System keystrokes generate system keystroke messages, like WM_SYSKEYDOWN and WM_SYSKEYUP. Nonsystem keystrokes generate nonsystem keystroke messages, like WM_KEYDOWN and WM_KEYUP.

A system keystroke message is produced when the user types a key in amalgamation with the ALT key or when the user types a key and the focus is NULL. If the focus is NULL, the keyboard event is transported to the active window. A system keystroke message has the WM_SYS prefix in the message name. A system keystroke message is used chiefly by the system rather than by an application. The system uses such a message to offer its incorporated keyboard interface to menus and to allow the user to manage which window is active. If a window procedure processes a system keyboard message, the window procedure should pass the message to the DefWindowProc function. Or else, all system operations that include the ALT key are disabled whenever that window has the keyboard focus.

The window procedure of the window that has the keyboard focus obtains all keystroke messages. Though, an application that reacts to keyboard input usually processes WM_KEYDOWN messages only.

Notes

When the window procedure obtains the WM_KEYDOWN message, it should inspect the virtual-key code that accompanies the message to find out how to process the keystroke. The virtual-key code is included in the *wParam*parameter of the message.

The *lParam* parameter of a keystroke message includes additional data regarding the keystroke that produced the message. The following table displays the additional keystroke data that is needed by the *lParam* parameter.

Data	Description
Context code	The value is 1 if the ALT key was pressed or 0 if the pressed key was released.
Previous key state	The value is 1 if the pressed key was down before or 0 if the pressed key was up before. The value is 1 for WM_KEYDOWN and WM_SYSKEYDOWN keystroke messages that were produced by automatic repeat functionality.
Repeat count	States the number of times that the keystroke was repeated as a result of the user holding down the key.
Scan code	Provides the hardware-dependent key scan code.
Transition state	The value is 1 if the key was released or if the key was pressed.

Usually, an application processes only the keystrokes that are produced by noncharacter keys.

Example: The following code example displays the window procedure framework that a usual application uses to obtain and process keystroke messages.

```
case  WM_KEYDOWN:
        switch  (wParam)
       {
          case VK_HOME:
             // Insert code here to process the HOME key
            // ...
            break;

          case VK_END:
             // Insert code here to process the END key
            // ...
            break;

          case VK_INSERT:
             // Insert code here to process the INS key
            // ...
            break;

          case VK_F2:
             // Insert code here to process the F2 key
            // ...
            break;
```

Notes

```
        case VK_LEFT:
           // Insert code here to process the LEFT ARROW key
          // ...
          break;

        case VK_RIGHT:
           // Insert code here to process the RIGHT ARROW key
          // ...
          break;

        case VK_UP:
           // Insert code here to process the UP ARROW key
          // ...
          break;

        case VK_DOWN:
           // Insert code here to process the DOWN ARROW key
          // ...
          break;

        case VK_DELETE:
           // Insert code here to process the DELETE key
          // ...
          break;

        default:
            // Insert code here to process other noncharacter keystrokes
          // ...
          break;
    }
```

Task Make distinction between WM_KEYDOWN and WM_KEYUP messages.

Self Assessment

Fill in the blank:

4. A system keystroke message is produced when the user types a key in amalgamation with the ALT key or when the user types a key and the focus is .....

Notes

7.4 WM_CHAR Message

WM_CHAR message is posted to the window with the keyboard focus when a WM_KEYDOWN message is converted by the Translate Message function.

The WM_CHAR message includes the character code of the key that was pressed.

```
#define WM_CHAR                    0x0102
```

7.4.1 Parameters

wParam: It is the character code of the key.

lParam: The repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag, as displayed in the following table.

Bits	Meaning
0-15	The repeat count for the existing message. The value is the number of times the keystroke is auto repeated as a consequence of the user holding down the key. If the keystroke is held long enough, numerous messages are sent. However, the repeat count is not cumulative.
16-23	The scan code. The value relies on the OEM.
24	Signifies whether the key is an extended key, like the right-hand ALT and CTRL keys that appear on an enhanced 101- or 102-key keyboard. The value is 1 if it is an extended key; or else, it is 0.
25-28	Reserved; do not use.
29	The context code. The value is 1 if the ALT key is held down while the key is pressed; or else, the value is 0.
30	The previous key state. The value is 1 if the key is down before the message is sent, or it is 0 if the key is up.
31	The transition state. The value is 1 if the key is being released, or it is 0 if the key is being pressed.

7.4.2 Return Value

An application should return zero if it processes this message.

Remarks

The **WM_CHAR** message utilizes Unicode Transformation Format (UTF)-16.

Since there is not necessarily a one-to-one correspondence among keys pressed and character messages produced, the information in the high-order word of the *lParam* parameter is usually not useful to applications. The information in the high-order word applies only to the most recent **WM_KEYDOWN** message that precedes the posting of the **WM_CHAR** message.

For improved 101- and 102-key keyboards, extensive keys are the right ALT and the right CTRL keys on the chief section of the keyboard; the INS, DEL, HOME, END, PAGE UP, PAGE DOWN and arrow keys in the clusters to the left of the numeric keypad; and the divide (/) and ENTER keys in the numeric keypad. Some other keyboards may sustain the extended-key bit in the *lParam* parameter.

The **WM_UNICHAR** message is the similar as **WM_CHAR**, except it utilizes UTF-32. It is intended to send or post Unicode characters to ANSI windows, and it can manage Unicode additional Plane characters.

Notes

Self Assessment

Fill in the blanks:

5. ..... message is posted to the window with the keyboard focus when a WM_KEYDOWN message is converted by the Translate Message function.
6. ..... is the character code of the key.

7.5 System Key Messages and Dead Characters

Some non-English keyboards enclose character keys that are not predictable to generate characters by themselves. Rather, they are used to add a diacritic to the character generated by the consequent keystroke. These keys are known as *dead keys*.

Example: The circumflex key on a German keyboard is an example of a dead key.

To enter the character including an "o" with a circumflex, a German user would type the circumflex key followed by the "o" key. The window with the keyboard focus would obtain the following series of messages:

```
WM_KEYDOWN
WM_DEADCHAR
WM_KEYUP
WM_KEYDOWN
WM_CHAR
WM_KEYUP
```

TranslateMessage produces the WM_DEADCHAR message when it processes the WM_KEYDOWN message from a dead key. Even though the *wParam* parameter of the WM_DEADCHAR message includes the character code of the diacritic for the dead key, an application usually overlooks the message. Rather, it processes the WM_CHAR message produced by the subsequent keystroke. The *wParam* parameter of the WM_CHAR message comprises the character code of the letter with the diacritic. If the subsequent keystroke produces a character that cannot be combined with a diacritic, Windows produces two WM_CHAR messages. The *wParam* parameter of the first comprises the character code of the diacritic; the *wParam*parameter of the second includes the character code of the subsequent character key.

The TranslateMessage function produces the WM_SYSDEADCHAR message when it processes the WM_SYSKEYDOWN message from a system dead key (a dead key that is pressed in amalgamation with the ALT key).

Did u know? An application in general overlooks the WM_SYSDEADCHAR message.

Self Assessment

Fill in the blank:

7. TranslateMessage produces the WM_DEADCHAR message when it processes the WM_KEYDOWN message from a ..... key.

Notes

7.6 Implementing a Simple Keyboard Interface

The .NET Framework Windows Form client is a huge platform for providing a rich, interactive and receptive user interface. The rich interface of a WinForm client, correctly constructed, can propose productivity advantages to the information worker far and above other standard architectures. There are many reasons for augmented productivity but one reason occurs from how simple it is to influence the keyboard in a WinForm client.

The Microsoft .NET Framework occurs packaged with all the building blocks to execute a keyboard interface in a WinForm application.

Remember that constancy is critical when implementing any interface, particularly a keyboard interface. Just as simple as it is to augment productivity with the keyboard, it is even simpler to reduce productivity by implementing a badly thought out and conflicting keyboard interface.

Example: If the F1 key pulls up a search box in one form of the application, and removes records in another form it can be harmful to the application all together. This is why the approaches discussed will concentrate on keeping the keyboard interface reliable while at the same time make the job of implementing the interface painless and competent for the .NET developer.

At the fundamental level capturing and taking action off of a keyboard command is easy and uncomplicated. By managing one of a few keyboard events (KeyDown, KeyUp, or KeyPress) on almost any WinForm .NET Control, keystrokes can be captured and suitable action taken. The severe problem with this is that the keyboard event would have to be managed individually for every control on the form to confirm that keyboard commands were always captured. It means that if a form had 5 buttons, 5 events would have to be managed and coded. This is neither practical nor competent for the .NET Developer and so causes the requirement to handle the keyboard events at the form level, in spite of what control has focus. The following discusses how to implement an event handler on the form that captures keystrokes.

Open Microsoft Visual Studio and generate a new Windows Form Application. Position 2 textbox controls and 2 button controls on the form. Open up the properties panel for the default form and rename the form to "MainForm". There is then a property on the form to set to true that is by default, false. The 'KeyPreview' property on the form causes keyboard events fired on controls on the form to be recorded with the form first. By managing the events at the form level it removes the requirement to handle them independently at the control level. Once the 'KeyPreview' property is set to true the subsequent step is to create the event handler on MainForm to manage the KeyDown event. Double-clicking the KeyDown event in the Properties window of MainForm will produce the event handler in code. Use the code block below to inhabit the KeyDown method.

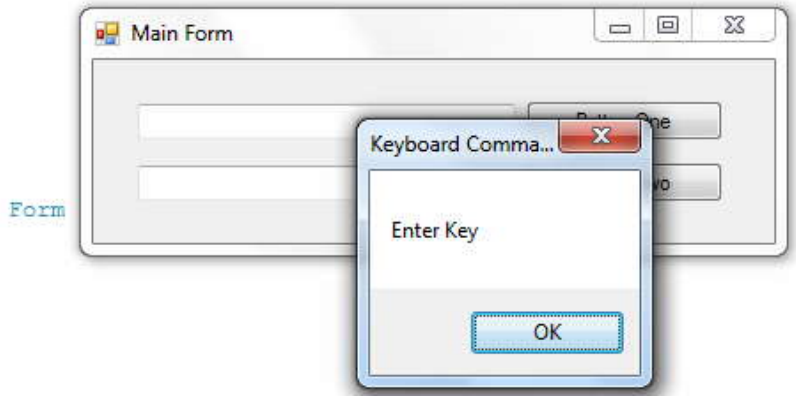
```
private void MainForm_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Escape)

MessageBox.Show("Escape Key", "Keyboard Command");

else if (e.KeyCode == Keys.F1)
```

Notes

```
MessageBox.Show("F1 Key", "Keyboard Command");

else if (e.KeyCode == Keys.Enter)

MessageBox.Show("Enter Key", "Keyboard Command");

}
```

The above code exhibits a message box with a message when the Escape, F1 or Enter key is pressed. Run the application and observe that regardless of what control has focus on the form, the KeyDown event on the form manages the keyboard command.

Figure 7.2



The KeyEventArgs utilized in the KeyDown event can also be used identify when modifiers are used (a grouping of keys, such as 'Ctrl+Alt') and even control if the keyboard command is passed down to the primary control.

The problem by means of this approach on each individual form is efficiency for the .NET Developer. By means of this approach in larger applications with many forms is not only incompetent but prone to bring in constancy issues. An application with number of forms will need the .NET Developer to individually setup and configure the keyboard events at a quantifiable impact to time and cost. This is where implementing an approach to expand .NET Controls and customizing events occurs in. By merging customized .NET Controls and inheritance the needed work on the Developer can be decreased while increasing the constancy in the way the keyboard interface is implemented.

There is another approach to capturing all keystrokes at the main form level. It includes implementing an IMessageFilter interface on the form level. While this is more effectual in some cases to capture all keystrokes it does have the potential of degrading performance as message filters are being added to the message pump for the application.

Self Assessment

Fill in the blank:

8. The '.....' property on the form causes keyboard events fired on controls on the form to be recorded with the form first.

Notes

7.7 Selecting a Stock Font

Windows comprises six stock fonts that are always obtainable. The fg_fontload() function states which stock font fg_print() uses for showing strings. The stock font identifiers and their numeric equivalents are:

```
OEM_FIXED_FONT (10)
ANSI_FIXED_FONT (11)
ANSI_VAR_FONT (12)
SYSTEM_FONT (13)
DEVICE_DEFAULT_FONT (14)
SYSTEM_FIXED_FONT (16)
```

Passing one of these identifiers or its numeric equivalent to fg_fontload() makes that font the existing font, meaning fg_print() will use it for exhibiting strings.

To select the system font, for example, call **fg_fontload(13)** or **fg_fontload(SYSTEM_FONT)**. By default, fg_print() utilizes the OEM_FIXED_FONT because this font most closely looks like the BIOS font used Fastgraph for DOS. We'll show an example program that uses fg_fontload() in the next section.

Self Assessment

Fill in the blank:

9. The ..... function states which stock font fg_print() uses for showing strings.

7.8 Using Logical Fonts

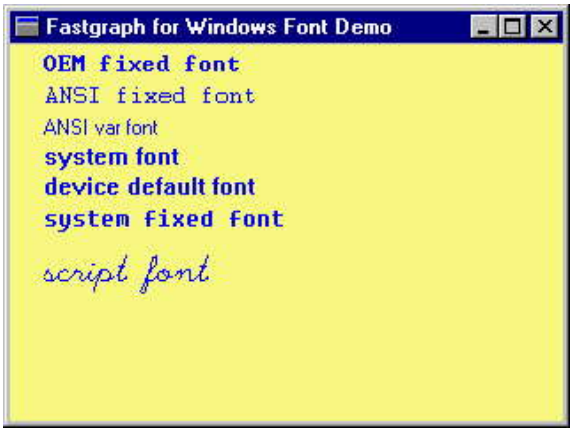
Windows allows you to create additional fonts relying on the stock fonts, or from character definitions in an external font file. Such fonts are known as logical fonts, and programs frequently use logical fonts to offer additional typefaces and dissimilar character sizes. The Windows API functions **CreateFont()** or **CreateFontIndirect()** generate logical fonts, and the fg_logfont() function allows you to use logical fonts in a Fastgraph program.

Example: Logical Fonts

Generating a logical font first includes defining the fields in a LOGFONT structure. In the Fontdemo example, we perform this at the end of the WM_CREATE handler, filling the LOGFONT structure with values to generate a 24x12 script font from one of the stock fonts. The WM_CREATE handler then passes this structure to the Windows API function **CreateFontIndirect()**, which returns a handle of type HFONT to the new logical font. We can then pass this font handle to fg_logfont() to turn on the font for fg_print() in the WM_PAINT handler.

Apart from showing how to create a logical font, the Fontdemo example displays how to use stock fonts and logical fonts in Fastgraph programs. Its WM_PAINT handler consecutively activates each of the six stock fonts and calls fg_print() to exhibit the font name by means of each font (strings are output directly to the client area). It then activates the logical font and again calls fg_print() to exhibit the text "script font" by means of the logical font. The result appears like this:

Notes



Notes We use fg_fontload() to activate a stock font, but we utilize fg_logfont() to turn on a logical font.

As displayed in the Fontdemo program's WM_DESTROY message handler, you should use the Windows API function **DeleteObject()** to remove any logical fonts you create. It is not essential to erase stock fonts.

Self Assessment

Fill in the blanks:

10. The Windows API functions CreateFont() or CreateFontIndirect() generate ..... fonts.
11. We use fg_fontload() to activate a stock font, but we utilize ..... to turn on a logical font.

7.9 Text Metric

Text Metric is a font structure contains basic information about a physical font.

```
typedef struct tagTEXTMETRIC {
   LONG tmHeight;
   LONG tmAscent;
   LONG tmDescent;
   LONG tmInternalLeading;
   LONG tmExternalLeading;
   LONG tmAveCharWidth;
   LONG tmMaxCharWidth;
   LONG tmWeight;
   LONG tmOverhang;
   LONG tmDigitizedAspectX;
```

Notes

```
   LONG  tmDigitizedAspectY;
   char  tmFirstChar;
   char  tmLastChar;
   char  tmDefaultChar;
   char  tmBreakChar;
   BYTE  tmItalic;
   BYTE  tmUnderlined;
   BYTE  tmStruckOut;
   BYTE  tmPitchAndFamily;
   BYTE  tmCharSet;
}  TEXTMETRIC;
```

7.9.1 Members

tmHeight

States the height (ascent descent) of characters.

tmAscent

States the ascent (units above the base line) of characters.

tmDescent

States the descent (units below the base line) of characters.

tmInternalLeading

States the amount of leading (space) inside the bounds set by the **tmHeight** member. Accent marks and other diacritical characters may happen in this area. The designer may set this member to zero.

tmExternalLeading

States the amount of extra leading (space) that the application adds among rows. Since this area is outside the font, it includes no marks and is not altered by text output calls in either OPAQUE or TRANSPARENT mode. The designer may set this member to zero.

tmAveCharWidth

States the average width of characters in the font (generally defined as the width of the letter x). This value does not comprise the overhang needed for bold or italic characters.

tmMaxCharWidth

States the width of the widest character in the font.

tmWeight

Specifies the weight of the font.

tmOverhang

Specifies the extra width per string that may be added to some synthesized fonts. When synthesizing some attributes, such as bold or italic, graphics device interface (GDI) or a device may have to add width to a string on both a per-character and per-string basis.

Notes

Example: GDI makes a string bold by expanding the spacing of each character and overstriking by an offset value; it italicizes a font by shearing the string. In either case, there is an overhang past the basic string. For bold strings, the overhang is the distance by which the overstrike is offset. For italic strings, the overhang is the amount the top of the font is sheared past the bottom of the font.

The **tmOverhang** member enables the application to determine how much of the character width returned by a GetTextExtentPoint32 function call on a single character is the actual character width and how much is the per-string extra width. The actual width is the extent minus the overhang.

tmDigitizedAspectX

Specifies the horizontal aspect of the device for which the font was designed.

tmDigitizedAspectY

Specifies the vertical aspect of the device for which the font was designed. The ratio of the **tmDigitizedAspectX** and **tmDigitizedAspectY** members is the aspect ratio of the device for which the font was designed.

tmFirstChar

Specifies the value of the first character defined in the font.

tmLastChar

Specifies the value of the last character defined in the font.

tmDefaultChar

Specifies the value of the character to be substituted for characters not in the font.

tmBreakChar

Specifies the value of the character that will be used to define word breaks for text justification.

tmItalic

Specifies an italic font if it is nonzero.

tmUnderlined

Specifies an underlined font if it is nonzero.

tmStruckOut

Specifies a strikeout font if it is nonzero.

tmPitchAndFamily

Specifies information about the pitch, the technology, and the family of a physical font.

Value	Description
TMPF_FIXED_PITCH	If this bit is set the font is a variable pitch font. If this bit is clear the font is a fixed pitch font. Observe that those meanings are the opposite of what the constant name implies.
TMPF_VECTOR	If this bit is set, the font is a vector font.
TMPF_TRUETYPE	If this bit is set, the font is a TrueType font.
TMPF_DEVICE	If this bit is set, the font is a device font.

Notes

The four low-order bits of this member state information regarding the pitch and the technology of the font. A constant is defined for each of the four bits.

An application should cautiously test for qualities encoded in these low-order bits, making no arbitrary assumptions.

Example: Apart from having their own bits set, TrueType and PostScript fonts set the TMPF_VECTOR bit. A monospace bitmap font has all of these low-order bits clear; a proportional bitmap font sets the TMPF_FIXED_PITCH bit. A Postscript printer device font sets the TMPF_DEVICE, TMPF_VECTOR, and TMPF_FIXED_PITCH bits.

The four high-order bits of **tmPitchAndFamily** assign the font's font family. An application can utilize the value 0xF0 and the bitwise AND operator to mask out the four low-order bits of **tmPitchAndFamily**, therefore receiving a value that can be directly compared with font family names to locate an identical match.

tmCharSet

States the character set of the font. The character set is one of the following values:

ANSI_CHARSET	DEFAULT_CHARSET
SYMBOL_CHARSET	SHIFTJIS_CHARSET
HANGUL_CHARSET	GB2312_CHARSET
CHINESEBIG5_CHARSET	OEM_CHARSET
JOHAB_CHARSET	HEBREW_CHARSET
ARABIC_CHARSET	GREEK_CHARSET
TURKISH_CHARSET	VIETNAMESE_CHARSET
THAI_CHARSET	EASTEUROPE_CHARSET
RUSSIAN_CHARSET	MAC_CHARSET
BALTIC_CHARSET	

Self Assessment

Fill in the blanks:

12. ..... is a font structure contains basic information about a physical font.

13. ..... states an underlined font if it is nonzero.

7.10 Putting Fonts to Work

7.10.1 Installing OpenType or TrueType Fonts in Windows

We suggest installing only one format — OpenType, TrueType, or PostScript — of a font. Installing two or more formats of the similar font may cause troubles when you attempt to use, view, or print the font.

1. Select **Start > Settings > Control Panel**

Notes In Windows XP, select **Start > Control Panel.**

Notes

2. Double-click the Fonts folder.
3. Select **File > Install New Font**.
4. Position the fonts you want to install.
   - In the **Drives** list, choose the drive and the folder containing the fonts you want to install.
   - In the **Folders** list, choose a folder that contains the fonts you want to install. (Ensure you have unzipped them first.) The fonts in the folder occur under List of Fonts
5. Choose the fonts to install. To choose more than one font, hold down the CTRL key and click each font.
6. To copy the fonts to the Fonts folder, ensure the **Copy fonts to the Fonts folder** check box is selected.

   Caution If installing fonts from a floppy disk or a CD-ROM, you should ensure this check box is chosen. Or else, to use the fonts in your applications, you always keep the disk in the disk drive.
7. Click **OK** to install the fonts.

7.10.2 Installing PostScript Type 1 Fonts in WindowsXP or 2000

PostScript Type 1 support is built into Windows XP and Windows 2000.

We suggest installing only one format — OpenType, TrueType, or PostScript — of a font. Installing two or more formats of the similar font may cause troubles when you attempt to use, view, or print the font.

1. Select **Start > Settings > Control Panel**

   Notes In Windows XP, choose **Start > Control Panel**
2. Double-click the Fonts folder.
3. Select **File > Install New Font**.
4. Locate the fonts you want to install.
   - In the **Drives** list, choose the drive and the folder containing the fonts you want to install.
   - In the **Folders** list, select a folder that contains the fonts you want to install. (Ensure you have unzipped them first.) The fonts in the folder occur under List of Fonts.
   - Choose the fonts to install. To select more than one font, hold down the CTRL key and click each font.
   - To copy the fonts to the Fonts folder, ensure the **Copy fonts to the Fonts folder** check box is selected.

Notes

If installing fonts from a floppy disk or a CD-ROM, you should ensure this check box is selected. Or else, to use the fonts in your applications, you must always keep the disk in the disk drive.

- Click **OK** to install the fonts.

Self Assessment

Fill in the blank:

14. To ..... the fonts to the Fonts folder, ensure the **Copy fonts to the Fonts folder** check box is selected.

7.11 Keyboard Accelerators

Accelerators are intimately associated to menus — both offer the user with access to an application's command set. Usually, users depend on an application's menus to study the command set and then switch over to using accelerators as they turn out to be more capable with the application. Accelerators offer faster, more direct access to commands than menus do. At a minimum, an application should supply accelerators for the more frequently used commands. Eve though accelerators usually generate commands that appear as menu items, they can also produce commands that have no corresponding menu items.

7.11.1 Accelerator Tables

An accelerator table includes an array of **ACCEL** structures, each defining an individual accelerator. Each **ACCEL** structure involves the following information:

- The accelerator's keystroke combination.
- The accelerator's identifier.
- Various flags. This comprises one that states whether the system is to offer visual feedback by stressing the corresponding menu item, if any, when the accelerator is used.

Caution To process accelerator keystrokes for a provided thread, the developer must call the **TranslateAccelerator** function in the message loop connected with the thread's message queue.

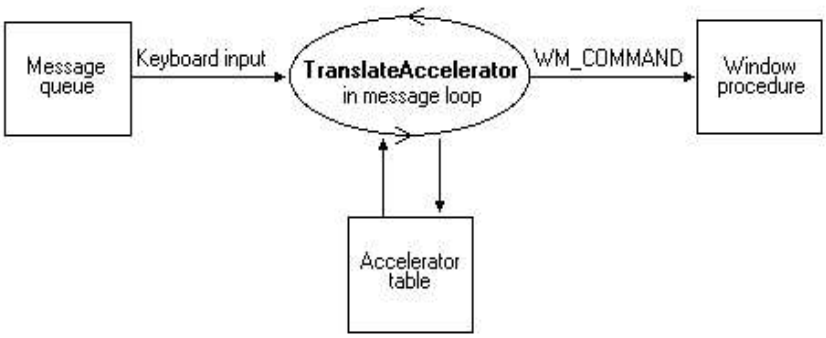
The **TranslateAccelerator** function monitors keyboard input to the message queue, verifying for key combinations that match an entry in the accelerator table. When **TranslateAccelerator** locates a match, it converts the keyboard input (that is, the **WM_KEYUP** and **WM_KEYDOWN** messages) into a **WM_COMMAND** or **WM_SYSCOMMAND** message and then sends the message to the window procedure of the particular window. The following illustration displays how accelerators are processed.

The **WM_COMMAND** message comprises the identifier of the accelerator that caused **TranslateAccelerator** to produce the message. The window procedure inspects the identifier to find out the source of the message and then processes the message consequently.

Accelerator tables survive at two different levels. The system sustains a single, system-wide accelerator table that applies to all applications. An application cannot amend the system accelerator table.

Notes

Figure 7.3



The system also sustains accelerator tables for each application. An application can define any number of accelerator tables for utilization with its own windows. A unique 32-bit handle (**HACCEL**) identifies each table. Though, only one accelerator table can be active at a time for a particular thread. The handle to the accelerator table passed to the **TranslateAccelerator** function identifies which accelerator table is active for a thread. The active accelerator table can be altered at any time by passing a dissimilar accelerator-table handle to **TranslateAccelerator**.

7.11.2 Accelerator Table Creation

Several steps are required to create an accelerator table for an application. First, a resource compiler is used to create accelerator-table resources and to add them to the application's executable file. At run time, the LoadAccelerators function is used to load the accelerator table into memory and retrieve the handle to the accelerator table. This handle is passed to the TranslateAccelerator function to activate the accelerator table.

An accelerator table can also be created for an application at run time by passing an array of ACCEL structures to the CreateAcceleratorTable function. This method supports user-defined accelerators in the application. Like the LoadAccelerators function, CreateAcceleratorTable returns an accelerator-table handle that can be passed to TranslateAccelerator to activate the accelerator table.

The system automatically destroys accelerator tables loaded by LoadAccelerators or created by CreateAcceleratorTable. However, an application can free resources while it is running by destroying accelerator tables no longer needed by calling the DestroyAcceleratorTable function.

An existing accelerator table can be copied and modified. The existing accelerator table is copied by using the CopyAcceleratorTable function. After the copy is modified, a handle to the new accelerator table is retrieved by calling CreateAcceleratorTable. Finally, the handle is passed to TranslateAccelerator to activate the new table.

7.11.3 Accelerator Keystroke Assignments

An ASCII character code or a virtual-key code can be used to define the accelerator. An ASCII character code makes the accelerator case sensitive. Thus, using the ASCII "C" character defines the accelerator as ALT+C rather than ALT+c. However, case-sensitive accelerators can be confusing to use. For example, the ALT+C accelerator will be generated if the CAPS LOCK key is down or if the SHIFT key is down, but not if both are down.

Typically, accelerators don't need to be case sensitive, so most applications use virtual-key codes for accelerators rather than ASCII character codes.

Notes

Avoid accelerators that conflict with an application's menu mnemonics, because the accelerator overrides the mnemonic, which can confuse the user. For more information about menu mnemonics, see Menus.

If an application defines an accelerator that is also defined in the system accelerator table, the application-defined accelerator overrides the system accelerator, but only within the context of the application. Avoid this practice, however, because it prevents the system accelerator from performing its standard role in the user interface. The system-wide accelerators are described in the following list:

ALT+ESC	Switches to the next application.
ALT+F4	Closes an application or a window.
ALT+HYPHEN	Opens the Window menu for a document window.
ALT+PRINT SCREEN	Copies an image in the active window onto the clipboard.
ALT+SPACEBAR	Opens the Window menu for the application's main window.
ALT+TAB	Switches to the next application.
CTRL+ESC	Switches to the Start menu.
CTRL+F4	Closes the active group or document window.
F1	Starts the application's help file, if one exists.
PRINT SCREEN	Copies an image on the screen onto the clipboard.

SHIFT+ALT+TAB Switches to the previous application. The user must press and hold down ALT+SHIFT while pressing TAB.

7.11.4 Accelerators and Menus

Using an accelerator is the same as choosing a menu item: Both actions cause the system to send a WM_COMMAND or WM_SYSCOMMAND message to the corresponding window procedure. The WM_COMMAND message includes an identifier that the window procedure examines to determine the source of the message. If an accelerator generated the WM_COMMAND message, the identifier is that of the accelerator. Similarly, if a menu item generated the WM_COMMAND message, the identifier is that of the menu item. Because an accelerator provides a shortcut for choosing a command from a menu, an application usually assigns the same identifier to the accelerator and the corresponding menu item.

An application processes an accelerator WM_COMMAND message in exactly the same way as the corresponding menu item WM_COMMAND message. However, the WM_COMMAND message contains a flag that specifies whether the message originated from an accelerator or a menu item, in case accelerators must be processed differently from their corresponding menu items. The WM_SYSCOMMAND message does not contain this flag.

The identifier determines whether an accelerator generates a WM_COMMAND or WM_SYSCOMMAND message. If the identifier has the same value as a menu item in the System menu, the accelerator generates a WM_SYSCOMMAND message. Otherwise, the accelerator generates a WM_COMMAND message.

If an accelerator has the same identifier as a menu item and the menu item is grayed or disabled, the accelerator is disabled and does not generate a WM_COMMAND or WM_SYSCOMMAND message. Also, an accelerator does not generate a command message if the corresponding window is minimized.

Notes

When the user uses an accelerator that corresponds to a menu item, the window procedure receives the WM_INITMENU and WM_INITMENUPOPUP messages as though the user had selected the menu item. For information about how to process these messages, see Menus.

An accelerator that corresponds to a menu item should be included in the text of the menu item.

7.11.5 UI State

Windows enables applications to hide or show various features in its UI. These settings are known as the UI state. The UI state includes the following settings:

- focus indicators (such as focus rectangles on buttons)
- keyboard accelerators (indicated by underlines in control labels)

A window can send messages to request a change in the UI state, can query the UI state, or enforce a certain state for its child windows. These messages are as follows.

Message	Description
WM_CHANGEUISTATE	Indicates that the UI state should change.
WM_QUERYUISTATE	Retrieves the UI state for a window.
WM_UPDATEUISTATE	Changes the UI state.

By default, all child windows of a top-level window are created with the same UI state as their parent.

The system handles the UI state for controls in dialog boxes. At dialog box creation, the system initializes the UI state accordingly. All child controls inherit this state. After the dialog box is created, the system monitors the user's keystrokes. If the UI state settings are hidden and the user navigates using the keyboard, the system updates the UI state. For example, if the user presses the Tab key to move the focus to the next control, the system calls WM_CHANGEUISTATE to make the focus indicators visible. If the user presses the Alt key, the system calls WM_CHANGEUISTATE to make the keyboard accelerators visible.

If a control supports navigation between the UI elements it contains, it can update its own UI state. The control can call WM_QUERYUISTATE to retrieve and cache the initial UI state. Whenever the control receives an WM_UPDATEUISTATE message, it can update its UI state and send a WM_CHANGEUISTATE message to its parent. Each window will continue to send the message to its parent until it reaches the top-level window. The top-level window sends the WM_UPDATEUISTATE message to the windows in the window tree. If a window does not pass on the WM_CHANGEUISTATE message, it will not reach the top-level window and the UI state will not be updated.

Self Assessment

Fill in the blanks:

15. An ..... table includes an array of **ACCEL** structures, each defining an individual accelerator.
16. The **WM_COMMAND** message comprises the identifier of the accelerator that caused ..... to produce the message.

Notes

7.12 Summary

- ANSI Character set is defined list of characters identified by the computer hardware and software.
- The three case conversion functions are LOWER, UPPER, and INITCAP and the five character manipulation functions are CONCAT. SUBSTR, LENGTH, INSTR, and LPAD.
- Keystroke messages handle window behavior and character messages verify the text that is exhibited in a window.
- The *lParam* parameter of a keystroke message includes additional data regarding the keystroke that produced the message.
- WM_CHAR message is posted to the window with the keyboard focus when a WM_KEYDOWN message is converted by the Translate Message function.
- TranslateMessage produces the WM_DEADCHAR message when it processes the WM_KEYDOWN message from a dead key.
- The fg_fontload() function states which stock font fg_print() uses for showing strings.
- Windows allows you to create additional fonts relying on the stock fonts, or from character definitions in an external font file. Such fonts are known as logical fonts, and programs frequently use logical fonts to offer additional typefaces and dissimilar character sizes.
- Text Metric is a font structure contains basic information about a physical font.
- The **TranslateAccelerator** function monitors keyboard input to the message queue, verifying for key combinations that match an entry in the accelerator table.

7.13 Keywords

ANSI Character Set: ANSI Character set is defined list of characters identified by the computer hardware and software.

Lparam: The *lParam* parameter of a keystroke message includes additional data regarding the keystroke that produced the message.

Text Metric: Text Metric is a font structure contains basic information about a physical font.

TranslateAccelerator: The **TranslateAccelerator** function monitors keyboard input to the message queue, verifying for key combinations that match an entry in the accelerator table.

TranslateMessage: TranslateMessage produces the WM_DEADCHAR message when it processes the WM_KEYDOWN message from a dead key.

WM_CHAR: WM_CHAR message is posted to the window with the keyboard focus when a WM_KEYDOWN message is converted by the Translate Message function.

7.14 Review Questions

1. What is ANSI Character Set? Also represent the ANSI Character Set table.
2. What are the different types of character functions? Illustrate.
3. Make distinction between CONCAT and INSTR function with example.
4. Explain the concept of processing keyboard messages.

Notes

5. Illustrate the function of WM_CHAR message. Also discuss the parameters.
6. TranslateMessage produces the WM_DEADCHAR message when it processes the WM_KEYDOWN message from a dead key. Comment.
7. Explain how to implement a Simple Keyboard Interface.
8. What are stock fonts? Also discuss using logical fonts with example.
9. Define text metric. Also illustrate the various members used in this structure.
10. Explain the concept of Keyboard Accelerators. Also illustrate the use of **TranslateAccelerator** function.

Answers: Self Assessment

1.	ANSI Character set	2.	LOWER
3.	INSTR	4.	INSTR
5.	WM_CHAR	6.	wParam
7.	dead	8.	KeyPreview
9.	fg_fontload()	10.	logical
11.	fg_logfont()	12.	Text Metric
13.	tmUnderlined	14.	copy
15.	accelerator	16.	TranslateAccelerator

7.15 Further Readings

Books

Brent E. Rector, *Win32 Programming,* Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

unicode.org/faq/font_keyboard.html

Notes

Unit 8: File I/O

CONTENTS

Objectives

Introduction

8.1 How Window Programs access Disk Files?

8.1.1 Opening

8.1.2 Reading, Writing, Closing: Disk Files

8.1.3 Read Disk File into Buffer

8.2 Creating File Selection Dialog Box

8.3 Creating a Text Editor

8.4 Summary

8.5 Keywords

8.6 Review Questions

8.7 Further Readings

Objectives

After studying this unit, you will be able to:

- Scan how windows programs access disk files
- Describe file selection dialog box
- Demonstrate text editor

Introduction

Now that you've added several programming statements to your Visual Basic language repertoire, you can learn about additional controls to add new features to your applications and to take advantage of some of the more powerful commands you now know. As you learn about new controls, your programming ability grows by leaps and bounds because your programs become richer in functionality and user interaction.

Almost every application in Microsoft Windows displays files, directories, and drives to the user in some fashion. In Visual Basic, files are displayed in a file list box, directories in a directory list box, and available drives in a drive list box. These controls will increase the magnitude of your application.

8.1 How Window Programs access Disk Files?

8.1.1 Opening

Sometimes installing new software in Windows can overwrite the file associations for other software. Graphics programs are notorious for this bad behavior. Fortunately, when it happens,

Notes

you can follow these steps to change the default file association so that double-clicking a file will open the program you choose instead of the last program you installed.

1. In Windows Explorer, navigate to a file of the type you want to associate (GIF, JPEG, etc.).
2. Click on its icon once to select it.
3. While holding the shift key down, right click on the icon. (Note: Holding the Shift key down is not necessary in Windows 2000 and Windows XP.)
4. In the pop-up menu, choose Open With... A box will open asking you to choose a program to open that file type. If you have a lot of software installed it could take a few moments for this box to appear.
5. Pick a program from the list.
6. If the program you need isn't in the list, choose [other] to navigate to another EXE file on your hard drive.
7. If you always want that program to open these types of files, put a checkmark in the box that says: Always use this program to open files of this type.

Notes

1. Make sure the program to which you associate a file type is capable of opening that type of file. Images must be opened in an image editor or viewer, text documents must be opened in a text editor or word processor, and so on.
2. You can also change file associations in Windows Explorer by going to View > Folder Options > File Types.

8.1.2 Reading, Writing, Closing: Disk Files

Writing and reading to disk files is accomplished in much the same way as reading and writing to the screen. The approach taken in C is to associate a stream with the file using fopen, and then use fprintf and fscanf for reading and writing. These behave exactly like printf and scanf except that they allow you to specify which stream they operate on (printf always uses stdout, while scanf uses stdin). When you have finished using the stream you should close it with fclose.

```
fopen
fopen opens a file and associates a stream with it. (Declared in stdio.h.)
        FILE *fopen(char *path, char *mode);
```

path is a string specifying the name of the file, while mode is a string indicating how the file is to be opened, typically either "r" to read from the file or "w" to write to it. If the file cannot be opened for some reason, fopen returns NULL.

```
        FILE *my_file;
/* .. */
my_file = fopen("datafile.txt", "r");  /* open datafile.txt for reading */
if (my_file == NULL) {
   fprintf(stderr, "Can't open datafile.txt for reading");
```

Notes

```
   exit(1);
}
```

The stream may be access via fprintf or fscanf, and should be subsequently closed with fclose. (In windows programs, HaltCL should be used instead of exit.)

```
fclose
```

fclose closes a stream previously opened with fopen. (Declared in stdio.h.)

```
       int fclose(FILE *stream);
```

A simple use of fclose might look like

```
       FILE *my_file;
/* .. */
my_file = fopen("datafile.txt", "r");
/* .. */
fclose(my_file);
```

```
fprintf
```

```
fprintf is the analogue of printf for use with arbitrary streams. (Declared
in stdio.h.)
```

```
       int fprintf(FILE *stream, const char *format, ... );
```

Example: To open a disk-file and write some text to it:

```
       FILE *my_file;
int i;
/* .. */
my_file=fopen("datafile.txt","w");
fprintf(my_file, "i=%d", i);
/* .. */
fclose(my_file);
```

The meaning of the second and subsequent arguments to fprintf are described in printf.

```
fscanf
```

```
fscanf is the analogue of scanf for use with arbitrary streams. (Declared
in stdio.h.)
```

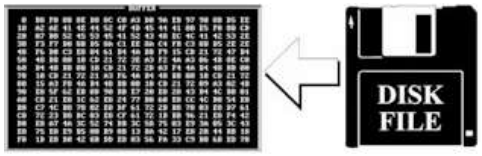
```
       int fscanf(FILE *stream, const char *format, ... );
```

Example: To open a disk-file and read some text from it:

```
       FILE *my_file;
int i;
/* .. */
my_file=fopen("datafile.txt","r");
fscanf(my_file, "%d", &i);   /*  read an integer from the file into i   */
/* .. */
fclose(my_file);
```

Notes

8.1.3 Read Disk File into Buffer



This command provides you with a variety of options for loading a binary, Intel Hex or Motorola S-record file into the buffer for viewing, editing or programming.

Command Options and File Load Summary

1. ***Normal (load at buffer base):*** This option is provided as a simple method for loading any file into the buffer. You need only provide a filename and the system will automatically determine the file type and load it into the buffer. Intel Hex and Motorola S-Record files are automatically loaded at the addresses specified in the file address records.
2. ***User defined load address:*** This option allows you to specify where in the buffer a binary file will be loaded. You may use this option to load multiple files into the buffer for editing or programming.
3. ***User defined base address:*** This option allows you to specify the base address used by the buffer. In essence this option allows you to redefine buffer address 0 to be any address within a 4 gigabyte address space (0-FFFFFFFF). With the buffer base redefined, you may load hex files with address records located anywhere within a 32 bit physical address range.
4. ***16/32 bit data path:*** This option allows you to load data for odd/even devices (16 bit data path) or 0, 1, 2, 3 devices (32 bit data path). This is also called byte split and shuffle. This option functions for both binary and hex type files.
5. ***Load sequential hex string:*** This option allows you to load a sequential string of hex characters into the buffer. This option originated from requests by users who needed to enter straight ASCII hex information without load addresses. Files of this type are usually created by a word processor or text editor.
6. ***File type:*** This option allows you to override the automatic file type setting normally used by the system and force processing of a specific file type (binary, Intel hex or S-record). This option is only used if the file format is too corrupt or altered for the system to automatically determine its format.
7. ***Recall buffer/file data summary:*** This option will redisplay the file load summary described at the top of this page.

Notes The command provides a load summary after a file has been processed. The load summary displays the number of bytes loaded and where in the buffer they were placed. If a hex file has been processed, the summary will display not only where the data was loaded, but also where hex data was encountered. If you attempt to load a hex file with addresses outside the buffer range, the system will show where the hex data was encountered even though no bytes will be loaded. This is extremely helpful as it tells you where the data actually needs to be placed. With the information from the summary, you may use Option 3 to redefine the buffer base address to allow your hex file to actually be loaded.

Notes

Self Assessment

Fill in the blanks:

1. fopen opens a file and associates a …… with it.
2. Path is a string specifying the name of the file, while …….. is a string indicating how the file is to be opened.
3. fscanf is the analogue of scanf for use with ……. streams.
4. …… option is provided as a simple method for loading any file into the buffer.
5. Load sequential hex string option allows you to load a sequential string of hex characters into the……..

8.2 Creating File Selection Dialog Box

In this section, we'll write code so that your menu items actually do something other than displaying message boxes. In other words, the Edit > Cut menu will really cut text, and the Edit > Paste menu will really paste text.

So open up the project you completed for the previous section. Comment out or delete any message box code.

We'll start with the File > Open menu.

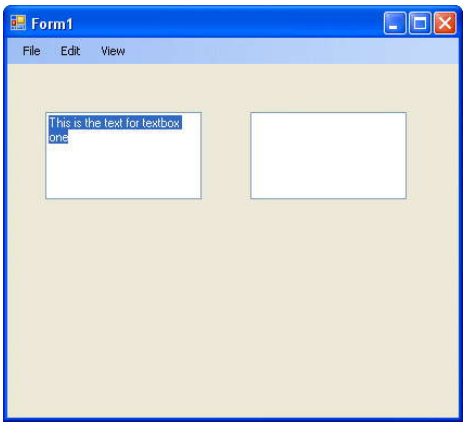
The Open File Dialogue Box

In most programmes, if you click the File menu, and select the Open item, a dialogue box is displayed. From the dialogue box, you can click on a file to select it, then click the Open button. The file you clicked on is then opened up. We'll see how to do that from our menu.

Caution Except, the file won't open yet - only the dialogue box will display, and then name of the chosen file.

First, place two textboxes on your form. In the properties box, locate the MultiLine property. Type some default text for the Text Property of textbox1. Change the Font size to 14 points.

Your form should now look something like this one:



Notes

Did u know? **What is the use of Open file dialog box?**

Use this dialog box to open an existing file from disk. You can also use this dialog box to open an already opened file using different language encoding options.

We'll work with these textboxes when we do the Edit menu. So let's leave them for now.

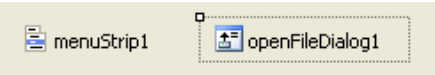
When we click on File > Open from our menu, we want the Open dialogue box to appear. This is fairly straightforward in VB.NET. In fact there is even a control for it!

Open up your toolbox, and locate the control called "OpenFileDialog". You might have to scroll down to see it. But you're looking for this:

Double click the control to add one to your project.

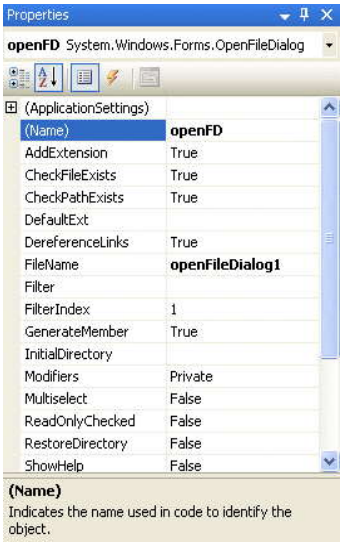
Notes But notice that the control doesn't get added to your form.

It gets added to the area at the bottom, next to your menu control:



The shaded area surrounding the control means that it is selected. If you look on your right, you'll see the properties that you can use with the control.

Click on the Name property and change the name to openFD. When you change the name in the properties box, the name of the control at the bottom will change:



Notes

We'll now write some code to manipulate the properties of our new control. So do the following:

- Access the code for your File > Open menu item. (To do this quickly, you can simply double click the Open item on your menu bar. Or, press F7 to access the Code View.)
- Click the name of your menu item from the left drop down box at the top of the code
- Then select the Click event from the drop down box to the right
- Your empty code should be this (the code below has underscore characters added, so that it can fit on this page):

```
Private Sub mnuOpen_Click(ByVal sender As Object, _
ByVal e As System.EventArgs) _
Handles mnuOpen.Click
End Sub
```

With you cursor flashing between the two lines of your code, add the following:

openFD.ShowDialog()

When you typed a full stop after the openFD, you probably saw a list box appear. You can just double click the ShowDialog() item to add it to your code.

Task To access this dialog box, select Open from the File menu and then choose File. Analyze.

But this method of the OpenFileDialog control does what you'd expect it to do: Shows the dialogue box. You can even test it out right now. Press F5 to run your program. Then click the Open item on your File menu. You should see an Open dialogue box display.

Return to the design environment, and we'll explore some more things you can do with this Dialogue box control.

The Initial Directory

You can set which directory the dialogue box should display when it appears. Instead of it displaying the contents of the "My Documents" folder, for example, you can have it display the contents of any folder. This done with the Initial Directory property. Amend your code to this:

```
openFD.InitialDirectory = "C:\"
openFD.ShowDialog()
```

Run your program again, and see the results in action. You should see the contents of the "C" folder on your hard drive (if you root folder is called something else, change the code above).

The Title Property

By default, the dialogue box will display the word "Open" as a caption at the top of your dialogue box. You can change this with the Title property. Add the line in Bold to your code:

```
openFD.InitialDirectory = "C:\"
openFD.Title = "Open a Text File"
openFD.ShowDialog()
```

Notes

Run your code again, and Click File > Open from your menu. You should see this at the top of the Open dialogue box:

Self Assessment

Fill in the blanks:

6. From the dialogue box, you can click on a file to select it, then click ..... button.
7. When we click on File > Open from our menu, we want the ..... box to appear.

8.3 Creating a Text Editor

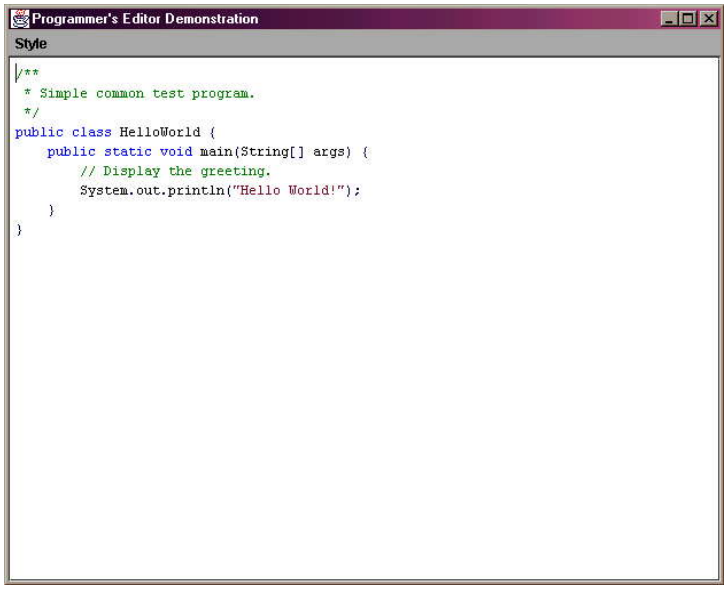
Despite initial appearances, writing a text editor is not a trivial task. Creating text editors takes programming experience and a firm basis in theory. The internet is littered with hundreds of unfinished or poorly working text editors. While this topic does not tell you everything you need to know to write your own text editor, you will learn how to add syntax highlighting and gain a sense of the overall complexity of undertaking such a project.

Did u know? **What you will need for creating text editor?**

If you haven't done so already, download the syntax highlighting package and the ProgrammerEditorDemo.java that comes with it. If you would like you may browse the source code for the demo.

Running the Demo

The demo is a very simple text editor:



The demo supports cut, copy, paste, and syntax highlighting, but little else. It doesn't even open and save files.

Notes

The Basics

Syntax Highlighting works like this: you give the text to the lexer, it goes through it and gives it back to you piece by piece and tells you what color to make each piece.

Streams and Readers

The syntax lexers accept your document through Streams and Readers. Fortunately it is very easy to turn just about anything into a Stream or a Reader. Java comes with many prebuilt classes for this purpose. A FileReader or a StringReader could be used. The demo uses a custom DocumentReader.

Tokens

The lexer returns Tokens. Tokens don't tell you the actual color that the text should be, but they do give you enough information to figure it out. The token contains such useful information as the type of text, a description of the text, and the position of the text in the file.

Example: Basic Example

```
JavaLexer syntaxLexer = new JavaLexer(new StringReader(myDocumentText));
Token t;
while ((t = syntaxLexer.getNextToken()) != null){
     // color the part of the document
     // to which the token refers here.
}
```

The Demo uses a look-up hashtable to get the color of the text based on the description from the token.

```
SimpleAttributeSet style;

style = new SimpleAttributeSet();
StyleConstants.setFontFamily(style, "Monospaced");
StyleConstants.setFontSize(style, 12);
StyleConstants.setBackground(style, Color.white);
StyleConstants.setForeground(style, Color.black);
StyleConstants.setBold(style, false);
StyleConstants.setItalic(style, false);
styles.put("body", style);

style = new SimpleAttributeSet();
StyleConstants.setFontFamily(style, "Monospaced");
StyleConstants.setFontSize(style, 12);
StyleConstants.setBackground(style, Color.white);
StyleConstants.setForeground(style, Color.blue);
```

Notes

```
StyleConstants.setBold(style, true);
StyleConstants.setItalic(style, false);
styles.put("tag", style);

...
```

The section of text in the document is then colored according to the style retrieved from the look-up table.

```
document.setCharacterAttributes(
      t.getCharBegin(),
      t.getCharEnd()-t.getCharBegin(),
      getStyle(t.getDescription()),
     true
);
```

Coloring Parts of the Document

The entire document is colored and it looks good in the editor. You might think that this is the end of the story. Sadly, its not. Editors are meant to edit documents. The documents change. The obvious approach is to re-color the document when the text changes. This may work for small documents, but as the document size gets larger it will quickly become unwieldy. For a 1000 line document, it could take as much as a few seconds to color the entire document. Waiting a few seconds each time a character is typed does not make for a good text editor.

Notes The trick is that not all of the document needs to be re-colored when something changes. But how much really needs to be re-colored? Not many editors do this part right. We have seen editors that re-color the previous three lines and the next three lines. That approach works most of the time, but it is pretty easy to fool.

Initial State

Every so often the syntax lexer returns to what are known as the initial state. At these times, the lexer returns a token and continues lexing as if it were at the beginning of the document again. Since the lexer acts as if it were at the beginning of the document from an initial state, the lexer could be restarted from this point without effecting the coloring of what comes afterwards. It can be determined from the last token returned if the lexer is in the initial state after returning that token.

So that solves half the problem. Just re-color the document from the last initial state. If the user is only going to append to the end of the document, this solves the problem. We can just keep track of the last initial state and re-color from there to the end of the document. But what if something in the middle of the document changes? We really need to keep track of *all* initial states so that we can restart the lexer from near anywhere in the document. Then we won't need to color the entire rest of the document either. If the lexer returns to an initial state at the same point that it returned to an initial state the last time, the rest of the document is already colored correctly.

Notes

Example: The demo keeps the list of initial states in a balanced tree. If desired the list could be included with the meta data of the document or stored in some other fashion. Care must be taken when looking at initial positions after start position. If text has been added or removed from the document, the positions after the addition/removal will have changed.

Only what is Visible

The initial coloring time for a document may become an issue. One way around this would be to only color what is visible on the screen. If the user scrolls, then more of the document will have to be colored as the user scrolls.

Another approach that is used by the demo is to start a separate thread to do the coloring. In this case, the document coloring happens in the background and the user may modify the document while that happens.

Self Assessment

Fill in the blanks:

8. The syntax lexers accept your document through Streams and ……...
9. If the lexer returns to an initial state at the same point that it returned to an initial state the last time, the rest of the document is already …….. correctly.
10. The internet is littered with hundreds of unfinished or poorly working …….

Caselet

Television anywhere, anytime…

What is the future of television? How is the structure of television programming going to change? Will people be watching TV rather than Internet TV? How will TV-on-demand affect the industry?

Listing a series of such questions, Bill Roedy, the former head of MTV Networks International, confesses that he does not know the answer to these common posers. He hastens to caution that if anyone tells you they know, you should not believe them. "Nobody knows for sure. The architecture of the business is changing and every media company is grappling with it. This kind of earthquake can destroy great companies but it also provides tremendous opportunities. There are huge shifts in the wind and the goal is to catch that wind and go with it," writes Roedy in 'What Makes Business Rock: Building the world's largest global networks' (www.wiley.com).

Delivery Systems have Changed

Looking back, the author notes that the basic concept behind all of the media world has not changed – viz. the ability to send pictures and sound through the air so countless people can see and hear them at precisely the same time. However, what has changed, and is continuing to change, are the delivery systems, the way content is distributed and the ability of a viewer to choose what and when they want to see, he adds.

Contd...

Notes

An apt quote cited in the book is of Judy McGrath, the former MTV CEO, that everybody who is making TV content now is thinking about Twitter, Facebook, and some sort of social media connection. The biggest question, according to her, is what kind of content would be successful on the widest variety of platforms. And the answer, as she sees it, is to either aim for the stars or aim for the cool, influential fringe. "The big things are getting bigger, the small things need to be cool and influential, and the middle, the average programming, that's over."

The World in your Pocket

Going forward the buzzword is mobile, avers Roedy. Reminding that the television industry used to brag that it could bring the world into your living room, which by itself was quite extraordinary, he says that now the industry can put the world in your pocket, by delivering content to a BlackBerry, iPod, iPad, and eventually every mobile device, sticking to the pervasive theme of television anywhere, anytime.

An example mentioned in the book is smart TV, that is, a television capable of accessing the Internet. "In the past TVs were sold as cable-ready, but in 2010, 21 per cent of the TVs sold to consumers were Internet-enabled. The technology is evolving and there are still difficulties to be resolved like ease of navigation and eliminating the keyboard, but in the past technology has been able to overcome every hurdle."

Foreseeing that gradually all TVs will evolve into a combination of television and computer, Roedy explains that in such a scenario Internet services and websites like Twitter, Netflix, Google TV, Apple TV, and Amazon's streaming service are all going to be available on the living room television, as well as on smartphones and tablets.

Insatiable Need for Content

On the challenge of distribution faced by content providers, the author observes that the traditional business model of providing content to the cable system operators and the direct-to-home operators has become complicated owing to the proliferation of distribution platforms, the only thing common being the insatiable need for content. Reminisces Roedy that only a few years ago many people were writing off content, believing that the ability of people to upload material to the Internet would result in a world of user-generated content. "That was the original appeal of YouTube. User-generated content is available now and some of it is very good."

Suggesting, for instance, that if you would like to find someone who sings like a young Beyoncé or a newer Beyoncé, there are sites that will lead you to her, the author underlines that the vast majority of the audience wants Beyoncé – not a younger or newer version, but the real thing. The lesson that he draws is that professionally produced storytelling remains by far the most popular programming across the entire spectrum of platforms, from cable TV to mobile phones. "The most-often viewed videos on YouTube, for example, are highly produced materials that either are pirated or licensed, a trend that I believe will continue."

Choice of Viewing Windows

In the author's opinion, the most serious challenge facing content providers is figuring out which distribution services in what form produce the best revenue stream. "The equation is what screens among all the possible distribution methods to license with how

Contd...

Notes

much content, and under what terms and conditions. On top of all that, you have to determine what kind of viewing windows will allow you to best protect the basic product, the channel."

Instructive is the example given in the book of the 2011 deal between Viacom and Hulu, a website that runs TV content for free to viewers after it has aired and profits from advertising, and Hulu Plus, which offers a greater variety of programming and charges subscription fees. "A key point in that deal was a 21-day window. Unique in that agreement was the provision that Hulu Plus would wait 21 days after Viacom's most popular shows, Jersey Shore for example, are initially broadcast before making them available online. Nobody yet knows if 21 days is the correct model, but it's just another step in this evolutionary process."

With too many unknowns in the uncharted territory of newer platforms, mistakes can happen by moving forward too quickly or by waiting too long, warns Roedy. A case he cites is of Starz, a collection of pay TV channels, which licensed its programming to Netflix in 2008 for three years for a total of $25 million, essentially giving it away. "In contrast, in 2010 the Viacom-led partnership with MGM and Lionsgate, Epix, licensed its 3,000-plus movie titles to Netflix for five years for almost a billion dollars."

Sustainability of Subscriber Base

What can be ominous to the cable TV industry is the threat posed by alternative delivery systems to the sustainability of subscriber base. The industry loses business when the consumer cancels his/her cable subscription to receive content through the Internet. Called 'cord cutting,' the impact of this phenomenon is evident from these numbers, from the US market: "It's not unusual for an American to be paying as much as $150 a month to the cable company for the media triple play, cable TV, a broadband connection, and a phone service. But by subscribing to Netflix for less than $10 monthly, paying separately for broadband for less than $50, and using an Internet phone operator, that same consumer can cut his or her costs by about half."

Estimates given in the book speak of 14 per cent of televisions connected to the Internet in 2010; and of the percentage rising to 38 by 2014. "In the second quarter of 2010 the cable industry lost 216,000 subscribers. They just went away. In the third quarter an additional 120,000 left."

It may be heartening to the cable industry that the price advantage enjoyed by the competition may eventually erode. "For example, Netflix' $25 million deal with Starz expires in 2012 – and renewing it will be very expensive, so Netflix' cost of content is going to rise rapidly, costs it will have to pass along to its customers. It's possible that Netflix one day will be as expensive as cable TV… But certainly, with advertising and subscription revenues of $150 billion the cable TV industry will do everything possible to protect its revenue stream."

Three Priorities

And the cable industry has not been complacent, one learns, what with the series of capability upgrades that have happened in the mature geographies, in the form of HD, TVR, wireless, and VoD. "In addition, cable operators are increasingly broadband connection providers: by the beginning of 2011, 54 per cent of Internet connections were provided by those companies."

Contd...

Notes

The book has a snatch of insight from Mike Fries, the president and CEO of John Malone's Liberty Global, the largest cable company outside the US, that 99 per cent of all television is still viewed on the living room TV set and that half the revenue of the industry today comes from IP services that did not exist a decade ago.

To take the pace of innovations to the next level, and to keep subscribers from cutting the cord, Fries prescribes three things: "Connect our content to other devices, including PCs, tablets, and smartphones; bring third-party online content and apps to the TV; and revolutionise the user interface and experience."

Educative read that can add whole new perspectives to your otherwise routine TV viewing.

8.4 Summary

- Sometimes installing new software in Windows can overwrite the file associations for other software. Graphics programs are notorious for this bad behavior.
- Fortunately, when it happens, you can follow these steps to change the default file association so that double-clicking a file will open the program you choose instead of the last program you installed.
- In this section, we'll write code so that your menu items actually do something other than displaying message boxes.
- In other words, the Edit > Cut menu will really cut text, and the Edit > Paste menu will really paste text.
- So open up the project you completed for the previous section. Comment out or delete any message box code.
- Despite initial appearances, writing a text editor is not a trivial task. Creating text editors takes programming experience and a firm basis in theory.
- The internet is littered with hundreds of unfinished or poorly working text editors.

8.5 Keywords

FTP: File Transfer Protocol

Tokens: The token contains such useful information as the type of text, a description of the text, and the position of the text in the file.

8.6 Review Questions

1. Sometimes installing new software in Windows can overwrite the file associations for other software. Explain.
2. The stream may be access via fprintf or fscanf, and should be subsequently closed with fclose. Discuss.
3. Intel Hex and Motorola S-Record files are automatically loaded at the addresses specified in the file address records. Examine.
4. In most programmes, if you click the File menu, and select the Open item, a dialogue box is displayed. Give Reasons.
5. The name of this location ("Personal" or "My Documents") depends on your operating system version. Discuss.

Notes

6. If you select multiple files, File Name displays each selected file within quotation marks. Comment.
7. The lexer returns a token and continues lexing as if it were at the beginning of the document again. Explain.
8. Tokens don't tell you the actual color that the text should be, but they do give you enough information to figure it out. Explain with examples.

Answers: Self Assessment

Fill in the blanks:

1.	Stream	2.	Mode
3.	Arbitrary	4.	Normal (load at buffer base)
5.	Buffer	6.	TheOpen
7.	Open dialogue	8.	Readers
9.	Colored	10.	Text editors

8.7 Further Readings

Books

Charles Petzold, *Programming Windows*, Microsoft Press

Feng Yuan, *Windows Graphics Programming: Win32 GDI and DirectDraw*, Prentice Hall Professional

Herbert Schildt, *Windows Programming: Annotated Archives*, Osborne/McGraw-Hill

William H. Murray, Chris H. Pappas, *Windows 3.1 Programming*, Osborne McGraw-Hill

Online links

http://en.wikipedia.org/wiki/Text_editor

http://en.wikipedia.org/wiki/Dialog_box

Notes

Unit 9: Child and Pop Up Windows

CONTENTS

Objectives

Introduction

9.1 Creating a Child Window

9.2 Sending Messages to Child Windows

9.2.1 Child Windows

9.2.2 Relationship to Parent Window

9.2.3 Messages

9.3 Fixed Child Windows

9.4 Pop Up Windows

9.5 Summary

9.6 Keywords

9.7 Review Questions

9.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand creating a child window
- Discuss sending messages to child window
- Discuss fixed child windows
- Understand pop up windows

Introduction

xWindow helps to maintain child (popup) windows, especially when you need multiple child windows and each window needs different features. You can then have different sets of links which will open in their own customized windows.

The window is opened the first time you call the object's load() method. When you call the object's load() method you pass it a URL. If the window is not open it is opened with the parameters you initially preset when creating the xWindow object and the page at the URL is then loaded into the window. If the window is already open then the page at the URL is loaded into the window. The window is then focused (brought to the top).

xWindow will work in almost any Javascript-enabled browser including NN4. xWindow is part of the X library.

9.1 Creating a Child Window

How to create a ChildWindow Login Popup for Windows Phone 7?

Notes

After struggling with a navigation issue in my app of when to show/go to/return from the Settings page that captured the user name and password, I decided to pop up a window on any page where and when credentials were necessary but not currently known. Since I knew I was going to add more credential-requiring pages in my app, using a popup allowed me to reduce my dependency on navigation silliness between the pages, and have a more encapsulated design.

This will show you how to pop up a ChildWindow on the app page to grab the user's login credentials. The control allows the old username and password to be passed into the control so that previous values can appear. The tab order/enter key switches from textbox to textbox to button.

The sample application included in this post just displays the results on the calling page.

Childwindow Control Reference

The LoginChildWindow inherits from System.Windows.Controls.ChildWindow so you must add that as a reference.

Loginchildwindow.xaml

```
<tk:ChildWindow x:Class= "LoginChildWindow.LoginChildWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

Notes

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:tk="clr
namespace:System.Windows.Controls;assembly=System.Windows.Controls"
mc:Ignorable="d"
VerticalAlignment="Top"
HorizontalAlignment="Left"
Title="Login"
BorderBrush="Black"
BorderThickness="2"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneAccentBrush}"
d:DesignHeight="256" d:DesignWidth="480"
HasCloseButton="false">


<Grid x:Name="LayoutRoot" Height="202" Background="{StaticResource
PhoneBackgroundBrush}">
<TextBlock HorizontalAlignment="Left" TextWrapping="Wrap" Text="User:
"VerticalAlignment="Top" Margin="20,43,0,0"/>
<TextBlock HorizontalAlignment="Left" TextWrapping="Wrap" Text="Password:"
VerticalAlignment="Top" Margin="20,104,0,0"/>
<TextBox x:Name="txtUserId" VerticalAlignment="Top" Margin="117,23,8,0"
Height="62" FontSize="18.667" TabIndex="1" KeyUp="txtUserId_KeyUp"/>
<PasswordBox x:Name="txtPassword" VerticalAlignment="Top"
Margin="117,85,8,0" Height="62" FontSize="18.667" TabIndex="2"
KeyUp="txtPassword_KeyUp" MouseLeftButtonUp="txtPassword_MouseLeftButtonUp"/
>


<Button x:Name="btnLogin" Content="Login" Margin="117,132,214,0"
d:LayoutOverrides="Width" Click="btnLogin_Click" FontSize="18.667"
BorderBrush="{StaticResource PhoneAccentBrush}" BorderThickness="1"
Foreground="{StaticResource PhoneForegroundBrush}"
Background="{StaticResource PhoneInactiveBrush}" Height="58"
VerticalAlignment="Top" FontFamily="Tahoma" TabIndex="3" />
<!—<Button x:Name="btnCancel" Content="Cancel" HorizontalAlignment="Left"
Margin="232,132,0,0" VerticalAlignment="Top" FontSize="18.667"
BorderBrush="{StaticResource PhoneAccentBrush}" BorderThickness="1"
Foreground="{StaticResource PhoneForegroundBrush}"
Background="{StaticResource PhoneInactiveBrush}" Height="58"
FontFamily="Tahoma" Click="btnCancel_Click" />—>
</Grid>
</tk:ChildWindow>
```

VerticalAlignment and HorizontalAlignment are set so that the window appears at the top of the screen. Without them, on my HD7, the on-screen keyboard overlays the child window and the user plays a game of tapping the control then the keyboard to get to the right place. I used system fonts so that the control will work with the phone's current settings. If your app uses custom settings, you will need to change these. The cancel button, which is usually next to the login button, has been commented out on purpose.

Notes

Notes In my app, the login credentials are vital and canceling makes no sense. The user can always back button or start button away from the app, if they choose. The child window's upper right corner cancel (icon of small x) is also removed via HasCloseButton="false" for the same reason.

Loginchildwindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace LoginChildWindow
{
    public partial class LoginChildWindow : ChildWindow
    {
        public string Login { get; set; }
        public string Password { get; set; }

        public LoginChildWindow()
        {
            InitializeComponent();
        }
        protected override void OnOpened()
        {
            base.OnOpened();

            this.txtUserId.Text = this.Login;
        }
        private void btnLogin_Click(object sender, RoutedEventArgs e)
        {
            // test values
```

Notes

```
                    if ((txtUserId.Text == null) || (txtPassword.Password ==
null))
                {
                    MessageBox.Show("WP7: Username & Password must be filled
                    in before logging on.");
                    //this.DialogResult = false;
                }
                        if ((txtUserId.Text.Trim() == string.Empty) ||
                        (txtPassword.Password.Trim() == string.Empty))
               {
                    MessageBox.Show("WP7: Username & Password must be filled
                    in before logging on.");
                    //this.DialogResult = false;
                }
                else if (txtUserId.Text.Trim().Length < 2)
                {
                    MessageBox.Show("WP7: Invalid username or password. Be
                    sure to use the WAZUp website login, not the Windows
                    Azure login.");
                    //this.DialogResult = false;
                }
                else
                {
                    // values are good so close this childwindow
                    this.Login = this.txtUserId.Text.Trim();
                    this.Password = this.txtPassword.Password.Trim();
                    this.DialogResult = true;
                }
            }
            //private void btnCancel_Click(object sender, RoutedEventArgse)
            //{
            //     this.DialogResult = false;
            //}
                private void txtUserId_KeyUp(object sender, KeyEventArgs e)
            {
                if (e.Key == Key.Enter)
                {
                    if (txtPassword.Password.Length == 0)
                    {
                        txtPassword.Focus();
                    }
                    else
                    {
```

Notes

```
                btnLogin_Click(sender, e);
            }
        }
    }
        private void txtPassword_KeyUp(object sender, KeyEventArgs e)
    {
         if (e.Key == Key.Enter)
        {
              btnLogin_Click(sender, e);
        }
    }
         private void txtPassword_MouseLeftButtonUp(object sender,
        MouseButtonEventArgs e)
    {
        btnLogin_Click(sender, e);


    }
  }
}
```

The validation routines are shallow. Please use them as a starting point for your own app requirements. The KeyUp and Mouse events are to make sure the enter button moves the cursor to the next logical place in the child window. It just saves the customer a step.

Main Page's Calling Code

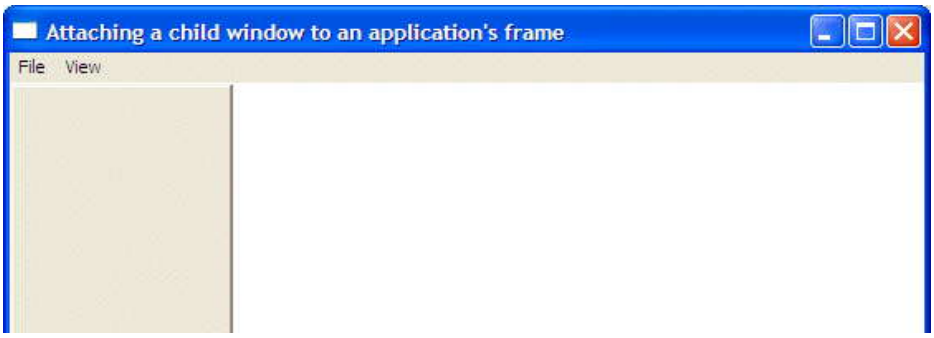
```
using System;
using System.Net;
using System.Windows;
using System.Windows.Input;
using Microsoft.Phone.Controls;

namespace LoginChildWindow
{
    public partial class MainPage : PhoneApplicationPage
  {
        public string Username { get; set; }
        public string Password { get; set; }

        // Constructor
        public MainPage()
        {
```

Notes

```
                Username = "OldUserName";
                 InitializeComponent();
                this.txtBlockUsername.Text = Username;
            }

              private void button1_Click(object sender, RoutedEventArgs e)
            {
                   LoginChildWindow loginWindow = new LoginChildWindow();
                   loginWindow.Login = Username;
                loginWindow.Closed += new EventHandler(OnLoginChildWindowShow);

                  loginWindow.Show();
            }
              private void OnLoginChildWindowShow(object sender, EventArgs e)
             {
                LoginChildWindow loginChildWindow = sender as LoginChildWindow;

                    if (loginChildWindow.DialogResult == true)
                 {
                         this.txtBlockUsername.Text = loginChildWindow.Login;
                        this.txtBlockPassword.Text = loginChildWindow.Password;
                      //;
                 }
             }
        }
    }
```

While I have included my version of System.Windows.Controls.dll in the app download, please use your own. I've grabbed several in the last six months from various locations and I don't know if mine is the right one.

Example:



Notes

This is an example of creating and attaching a child window, like a toolbox, to the main frame of an application.

```
Resource  Header

#define  IDD_TOOLBOX_DLG                    101
#define  IDR_MAIN_MENU                      102
#define  IDM_FILE_EXIT                      40001
#define  IDM_VIEW_TOOLBOX                   40002
Resource  Script

#include  "resource.h"

/////////////////////////////////////////////////////////////////////////////
////
//
//  Dialog
//

IDD_TOOLBOX_DLG  DIALOG  DISCARDABLE   0,  0,  86,  249
STYLE  DS_MODALFRAME  |  WS_CHILD
FONT  8,  "MS  Sans  Serif"
BEGIN
END

/////////////////////////////////////////////////////////////////////////////
////
//
//  Menu
//

IDR_MAIN_MENU  MENU  DISCARDABLE
BEGIN
     POPUP  "&File"
     BEGIN
         MENUITEM  "E&xit",                          IDM_FILE_EXIT
     END
     POPUP  "&View"
     BEGIN
         MENUITEM  "&Toolbox",                       IDM_VIEW_TOOLBOX,  CHECKED
     END
```

Notes

```
END

////////////////////////////////////////////////////////////////////////
////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
    IDM_FILE_EXIT            "Closes the application"
    IDM_VIEW_TOOLBOX         "Toggles the presence and disappearance of the
                            toolbox\nShow/Hide Toolbox"
END

#endif     // English (U.S.) resources
////////////////////////////////////////////////////////////////////////
////
Source Code

#include <windows.h>
#include "resource.h"

HINSTANCE hInst;
LPTSTR strAppName = "WndFrame";
LPTSTR WndName     = "Attaching a child window to an application's frame";
LPTSTR strToolbox = "WndFloater";

HWND    hWndMainFrame, hWndToolbox;

LRESULT CALLBACK MainWndProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM
lParam);
LRESULT CALLBACK ToolboxProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM
lParam);
//-------------------------------------
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                       LPSTR lpCmdLine, int nCmdShow)
{
    MSG   msg;
    RECT rect;
      WNDCLASSEX   WndClsEx;
```

Notes

```
    WndClsEx.cbSize           = sizeof(WNDCLASSEX);
    WndClsEx.style            = CS_HREDRAW | CS_VREDRAW;
    WndClsEx.lpfnWndProc      = MainWndProc;
    WndClsEx.cbClsExtra       = 0;
    WndClsEx.cbWndExtra       = 0;
    WndClsEx.hIcon            = LoadIcon(NULL, IDI_APPLICATION);
    WndClsEx.hCursor          = LoadCursor(NULL, IDC_ARROW);
    WndClsEx.hbrBackground                                    =
static_cast<HBRUSH>(GetStockObject(WHITE_BRUSH));
    WndClsEx.lpszMenuName   =   MAKEINTRESOURCE(IDR_MAIN_MENU);
    WndClsEx.lpszClassName  = strAppName;
    WndClsEx.hInstance       = hInstance;
    WndClsEx.hIconSm          = LoadIcon(NULL, IDI_APPLICATION);

    if (!RegisterClassEx(&WndClsEx))
          return (FALSE);

  hInst = hInstance;

   hWndMainFrame = CreateWindow(strAppName,
                                WndName,
                                WS_OVERLAPPEDWINDOW,
                                CW_USEDEFAULT,
                                 CW_USEDEFAULT,
                                 CW_USEDEFAULT,
                                 CW_USEDEFAULT,
                                NULL,
                                NULL,
                                 hInstance,
                                NULL);

  if( !hWndMainFrame )
    return (FALSE);


  // Create a child window based on the available dialog box
   hWndToolbox = CreateDialog(hInst,
                                  MAKEINTRESOURCE(IDD_TOOLBOX_DLG),
                        hWndMainFrame,
                        (DLGPROC)ToolboxProc);
```

Notes

```
        ShowWindow (hWndToolbox, SW_SHOW);
        ShowWindow(hWndMainFrame, nCmdShow);

        while (GetMessage(&msg,NULL, 0,0))
    {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
     }

     return 0;
}
//-------------------------------------
LRESULT CALLBACK ToolboxProc(HWND hWndDlg, UINT Msg, WPARAM wParam, LPARAM
lParam)
{
       switch(Msg)
       {
       case WM_INITDIALOG:
              return TRUE;
       }

       return FALSE;
}
//-------------------------------------
LRESULT CALLBACK MainWndProc(HWND hWnd, UINT Msg,
                             WPARAM wParam, LPARAM lParam)
{
       HMENU hMenu;
       RECT rctMainWnd, rctToolbox;
       UINT ToolboxMenuState;

       switch(Msg)
       {
       case WM_COMMAND:
              switch(LOWORD(wParam))
              {
              case IDM_VIEW_TOOLBOX:
                     hMenu = GetMenu(hWndMainFrame);
                     ToolboxMenuState = GetMenuState(hMenu,
                     IDM_VIEW_TOOLBOX, MF_BYCOMMAND);
```

Notes

```
            if( LOBYTE(ToolboxMenuState) & MF_CHECKED )
            {
                CheckMenuItem(hMenu, IDM_VIEW_TOOLBOX,
                MF_BYCOMMAND | MF_UNCHECKED);
                ShowWindow(hWndToolbox, SW_HIDE);
            }
            else
            {
                CheckMenuItem(hMenu, IDM_VIEW_TOOLBOX,
                MF_BYCOMMAND | MF_CHECKED);
                ShowWindow(hWndToolbox, SW_SHOW);
            }
            break;

        case IDM_FILE_EXIT:
            PostQuitMessage(WM_QUIT);
            return 0;
        };
        break;

    case WM_SIZE:
        GetClientRect(hWndMainFrame, &rctMainWnd);
        GetWindowRect(hWndToolbox, &rctToolbox);

        SetWindowPos(hWndToolbox,
                     HWND_TOP,
                     rctMainWnd.left,
                     rctMainWnd.top,
                     rctToolbox.right - rctToolbox.left,
                     rctMainWnd.bottom,
                     SWP_NOACTIVATE | SWP_NOOWNERZORDER);

        break;

    case WM_DESTROY:
        PostQuitMessage(WM_QUIT);
        break;

    default:
        return DefWindowProc(hWnd, Msg, wParam, lParam);
    }
```

Notes

```
        return 0;
}
//-----------------------------------------
```

Self Assessment

Fill in the blanks:

1. ..... helps to maintain child (popup) windows, especially when you need multiple child windows and each window needs different features.
2. When you call the object's load() method you pass it a .....
3. The ..... inherits from System.Windows.Controls.ChildWindow so you must add that as a reference

9.2 Sending Messages to Child Windows

9.2.1 Child Windows

A *child window* has the WS_CHILD style and is confined to the client area of its parent window. An application typically uses child windows to divide the client area of a parent window into functional areas. You create a child window by specifying the WS_CHILD style in the CreateWindowEx function.

A child window must have a parent window. The parent window can be an overlapped window, a pop-up window, or even another child window. You specify the parent window when you call CreateWindowEx.

Did u know? If you specify the WS_CHILD style in CreateWindowEx but do not specify a parent window, the system does not create the window.

9.2.2 Relationship to Parent Window

An application can change the parent window of an existing child window by calling the SetParent function. In this case, the system removes the child window from the client area of the old parent window and moves it to the client area of the new parent window. If SetParent specifies a NULL handle, the desktop window becomes the new parent window. In this case, the child window is drawn on the desktop, outside the borders of any other window. The GetParent function retrieves a handle to a child window's parent window.

The parent window relinquishes a portion of its client area to a child window, and the child window receives all input from this area. The window class need not be the same for each of the child windows of the parent window. This means that an application can fill a parent window with child windows that look different and carry out different tasks.

Example: A dialog box can contain many types of controls, each one a child window that accepts different types of data from the user.

Notes

Task Make distinction between child window and parent window.

9.2.3 Messages

The system passes a child window's input messages directly to the child window; the messages are not passed through the parent window. The only exception is if the child window has been disabled by the EnableWindow function. In this case, the system passes any input messages that would have gone to the child window to the parent window instead. This permits the parent window to examine the input messages and enable the child window, if necessary.

A child window can have a unique integer identifier. Child window identifiers are important when working with control windows. An application directs a control's activity by sending it messages. The application uses the control's child window identifier to direct the messages to the control. In addition, a control sends notification messages to its parent window. A notification message includes the control's child window identifier, which the parent uses to identify which control sent the message.

Notes An application specifies the child-window identifier for other types of child windows by setting the hMenu parameter of the CreateWindowEx function to a value rather than a menu handle.

Layered Windows

Using a layered window can significantly improve performance and visual effects for a window that has a complex shape, animates its shape, or wishes to use alpha blending effects. The system automatically composes and repaints layered windows and the windows of underlying applications. As a result, layered windows are rendered smoothly, without the flickering typical of complex window regions. In addition, layered windows can be partially translucent, that is, alpha-blended.

To create a layered window, specify the WS_EX_LAYERED extended window style when calling the CreateWindowEx function, or call the SetWindowLong function to set WS_EX_LAYERED after the window has been created. After the CreateWindowEx call, the layered window will not become visible until the SetLayeredWindowAttributes or UpdateLayeredWindow function has been called for this window.

Caution WS_EX_LAYERED cannot be used for child windows.

To set the opacity level or the transparency color key for a given layered window, call SetLayeredWindowAttributes. After the call, the system may still ask the window to paint when the window is shown or resized. However, because the system stores the image of a layered window, the system will not ask the window to paint if parts of it are revealed as a result of relative window moves on the desktop. Legacy applications do not need to restructure their painting code if they want to add translucency or transparency effects for a window, because the

Notes

system redirects the painting of windows that called SetLayeredWindowAttributes into off-screen memory and recomposes it to achieve the desired effect. For faster and more efficient animation or if per-pixel alpha is needed, call UpdateLayeredWindow. UpdateLayeredWindow should be used primarily when the application must directly supply the shape and content of a layered window, without using the redirection mechanism the system provides through SetLayeredWindowAttributes. In addition, using UpdateLayeredWindow directly uses memory more efficiently, because the system does not need the additional memory required for storing the image of the redirected window. For maximum efficiency in animating windows, call UpdateLayeredWindow to change the position and the size of a layered window.

Caution After SetLayeredWindow Attributes has been called, subsequent UpdateLayeredWindow calls will fail until the layering style bit is cleared and set again.

Hit testing of a layered window is based on the shape and transparency of the window. This means that the areas of the window that are color-keyed or whose alpha value is zero will let the mouse messages through. However, if the layered window has the WS_EX_TRANSPARENT extended window style, the shape of the layered window will be ignored and the mouse events will be passed to other windows underneath the layered window.

Message-Only Windows

A message-only window enables you to send and receive messages. It is not visible, has no z-order, cannot be enumerated, and does not receive broadcast messages. The window simply dispatches messages.

To create a message-only window, specify the HWND_MESSAGE constant or a handle to an existing message-only window in the hWndParentparameter of the CreateWindowEx function. You can also change an existing window to a message-only window by specifying HWND_MESSAGE in the hWndNewParent parameter of the SetParent function.

To find message-only windows, specify HWND_MESSAGE in the hwndParent parameter of the FindWindowEx function. In addition, FindWindowEx searches message-only windows as well as top-level windows if both the hwndParent and hwndChildAfter parameters are NULL.

Task Illustrate the function of message-only window.

Self Assessment

Fill in the blanks:

4. A *child window* has the ..... style and is confined to the client area of its parent window.
5. An application can change the parent window of an existing child window by calling the ..... function.
6. An application directs a control's activity by sending it .....
7. A ..... message includes the control's child window identifier, which the parent uses to identify which control sent the message.

Notes

8. Using a ..... window can significantly improve performance and visual effects for a window that has a complex shape, animates its shape, or wishes to use alpha blending effects.
9. To set the opacity level or the transparency color key for a given layered window, call ..... Attributes.
10. A ..... window enables you to send and receive messages. It is not visible, has no z-order, cannot be enumerated, and does not receive broadcast messages.
11. To find message-only windows, specify ..... in the hwndParent parameter of the FindWindowEx function.

9.3 Fixed Child Windows

[FIXED] Window that opens child Window - Menu displayed behind child

If you have a main Window which opens a child Window and that child Window has a Component which displays a menu, the menu will appear behind the child Window the first time it is displayed.

To test, run the code below, click the button, then click the date field trigger. Do not move the child window in between steps as this will probably alter the child window z-index.

Code:

```
public static void testPopupBug()
    {
        final Window w = new Window();
        w.setSize(100, 100);

        // On button click, display a new window with a simple date field
        Button b = new Button("Open new Window with date field");
         b.addSelectionListener(new SelectionListener<ButtonEvent>() {
            public void componentSelected(ButtonEvent ce)
          {
              final Window popup = new Window();
              popup.setSize(150, 150);

               final LayoutContainer c = new LayoutContainer();
              c.add(new DateField());
             popup.add(c);
             popup.show();
          }
       });

       w.add(b);
```

Notes

```
        // Display the main window after a short while
       new Timer()
       {
            @Override
            public void run()
           {
                w.show();
           }

        }.schedule(500);
    }
```

Self Assessment

Fill in the blank:

12. ..... Window opens child Window - Menu displayed behind child.

9.4 Pop Up Windows

A window that suddenly appears (pops up) when you select an option with a mouse or press a special function key. Usually, the pop-up window contains a menu of commands and stays on the screen only until you select one of the commands. It then disappears.

A special kind of pop-up window is a *pull-down menu*, which appears just below the item you selected, as if you had pulled it down.

A popup window is a web browser window that is smaller than standard windows and without some of the standard features such as tool bars or status bars. For instance, this link opens a medium-sized popup window.

Did u know? Popup windows (aka popups) are popular for small sidebar-style pages that are digressions from the main page.

Popups are one of the trickiest effects in web development. More than one web developer has been reduced to tears trying to get popups to work correctly. Furthermore, some irresponsible popup techniques have made many web pages handicapped and search engine inaccessible.

This topic will walk you step-by-step through creating popup windows, including giving you a complete set of copy-and-paste JavaScript code. We'll start with a basic example, showing the main pieces to a popup. Then we'll show the techniques for targeting a link inside the popup back to the main page. Finally we'll work through the many parameters for the open() command that adds features to your popups.

Example: Use the Dreamweaver behaviors panel to create small pop-up windows for ads or news!

This behavior helps to create pop-up ads, small browser windows etc. If you don't have Dreamweaver we have provided the code that you can cut 'n' paste and use in your web page.

Notes

'Open Browser Window' Behavior

This behavior allows you to open a browser window in any size you specify. E.g. Click here to see a sample.

Code for the Function: Cut 'n' Paste Code

```
<script language="JavaScript">
<!—
function MM_openBrWindow(theURL,winName,features) { //v2.0
window.open(theURL,winName,features);
}
//—>
</script>
```

Code for the Links: Cut 'n' Paste Code

```
<a href="javascript:;" onClick="MM_openBrWindow('/website_templates/business1/preview.htm','template','width=780,height=550')">Click Here </a>
```

The above code is taken from Macromedia Dreamweaver 4.0

Learn how to Open a New browser Window in Dreamweaver

1. Open the behaviors panel by clicking on Windows/Behaviors.
2. Select the text that you would like to link to the new browser window.
3. Click on the '+' symbol in the behaviors panel.
4. Click on 'Open Browser Window'
5. In the 'Open browser Window' pop-up-window select the URL and specify the width and the height of the window. You can also choose if you want a status bar or a scroll bar etc.
6. Click on OK
7. In the behaviors panel under the Events Column you can choose onClick if you want the browser window to open only when the link is clicked.

Self Assessment

Fill in the blanks:

13. A window that suddenly appears (pops up) when you select an option with a mouse or press a special function key is called a ..... window.
14. A special kind of pop-up window is a ..... *menu*, which appears just below the item you selected, as if you had pulled it down.
15. A popup window is a ..... window that is smaller than standard windows and without some of the standard features such as tool bars or status bars.

9.5 Summary

- xWindow helps to maintain child (popup) windows, especially when you need multiple child windows and each window needs different features.

Notes

- The LoginChildWindow inherits from System.Windows.Controls.ChildWindow so you must add that as a reference
- A *child window* has the WS_CHILD style and is confined to the client area of its parent window.
- An application can change the parent window of an existing child window by calling the SetParent function.
- The system passes a child window's input messages directly to the child window; the messages are not passed through the parent window.
- A notification message includes the control's child window identifier, which the parent uses to identify which control sent the message.
- Using a layered window can significantly improve performance and visual effects for a window that has a complex shape, animates its shape, or wishes to use alpha blending effects.
- A window that suddenly appears (pops up) when you select an option with a mouse or press a special function key is called a pop-up window.

9.6 Keywords

Child Window: A *child window* has the WS_CHILD style and is confined to the client area of its parent window.

Notification Message: A notification message includes the control's child window identifier, which the parent uses to identify which control sent the message.

Pop-up Window: A window that suddenly appears (pops up) when you select an option with a mouse or press a special function key is called a pop-up window.

XWindow: xWindow helps to maintain child (popup) windows, especially when you need multiple child windows and each window needs different features.

9.7 Review Questions

1. What do you mean by 'Child windows'? Illustrate its uses.
2. Can a child window be visible out of the parent's client area? Enlighten.
3. What will happen to the child Windows, if their parent is destroyed?
4. Illustrate the concept of fixed child windows.
5. What do you mean by Popup Window? Explain.
6. Which window style is used to create a popup window? Illustrate.
7. Where should the message be passed if it is not processed by the message processing logic in a message function for a child or a popup window?
8. What do you know about WM_USER message?
9. Why we use the Get Parent () function? Explain.
10. Illustrate the steps to create a child window with example.

Notes

Answers: Self Assessment

1. xWindow
2. url
3. LoginChildWindow
4. WS_CHILD
5. SetParent
6. messages
7. notification
8. layered
9. SetLayeredWindow
10. message-only
11. HWND_MESSAGE
12. [FIXED]
13. pop up
14. pull-down
15. web browser

9.8 Further Readings

Books

Brent E. Rector, *Win32 Programming,* Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

cross-browser.com/x/examples/xwindow.php

Notes

Unit 10: Menus

CONTENTS

Objectives

Introduction

10.1 Creating Menus

10.2 Menu defined as Resource Data

10.3 Creating a Menu using the Borland Resource Workshop

10.4 Complex Menu

10.4.1 Standard Functionality of Shortcut Keys

10.5 Creating a Menu as a Program Operates

10.6 Creating Menu Containing Bitmaps and the System Menu

10.6.1 Creating Menu Containing Bitmaps

10.6.2 The System Menu

10.7 Summary

10.8 Keywords

10.9 Review Questions

10.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of creating menus
- Illustrate Menu defined as resource data
- Discuss creating a menu using the borland resource workshop
- Understand the concept of complex menu
- Recognize how to creating menu containing bitmaps
- Discuss system menu

Introduction

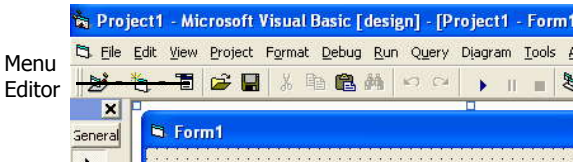
Menus are a very important element of GUIs. We see menus and we use them in practically every application, whether MS-Word or MS-Excel or Photoshop or even VB. In this unit, we will focus on creating menus using VB.

10.1 Creating Menus

For getting started with menus, we get going with the Menu Editor. This tool helps us to effortlessly create a menu interface for our applications, that too in no time. To invoke the menu editor, we either:

Notes

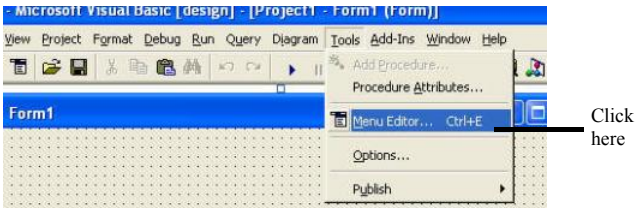
1. Click on the toolbar icon for the menu editor



Or

2. On the menu bar, click on 'Tools'

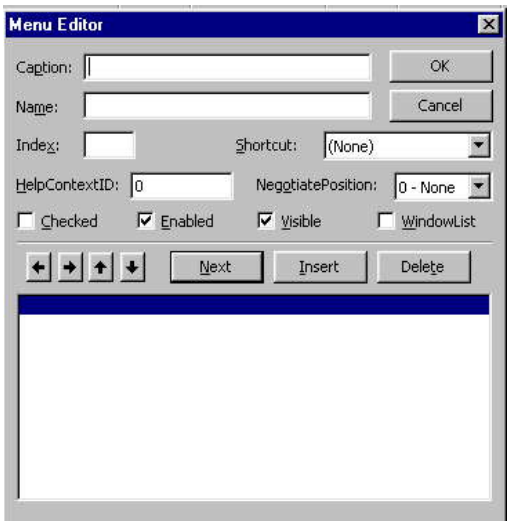
Menu Editor



Or

3. Press 'Ctrl + E' (the short-cut for the menu editor).

Whichever option we choose, we get the Menu Editor, shown here:



This is how we use the Menu Editor:

- In the topmost textbox 'Caption' of the figure, we type the 'Caption' of the menu, as the user will be seeing it.
- In the next textbox 'Name' of the figure, we type the 'Name' of the menu, as the computer will be seeing it.
- 'Caption' and 'Name' mean the same as we know; 'Caption' is the visible portion, 'Name' is for coding of the application.
- The 'Index' specifies the index (or subscript) for the menu. This is required if we are creating a menu array (an array of menus).

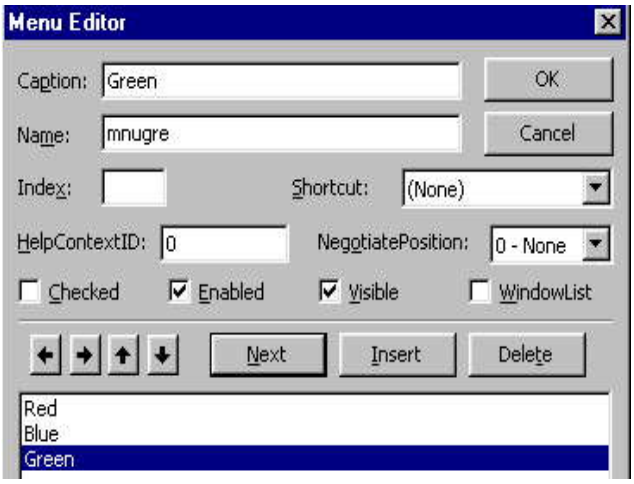
Notes

- The 'Enabled' and 'Visible' checkboxes are 'checked' by default, which means that the menu will be both visible and enabled at the time of creation.
- The 'Shortcut' option gives us a drop-down list for assigning shortcut keys to our menus (more on this a little later).

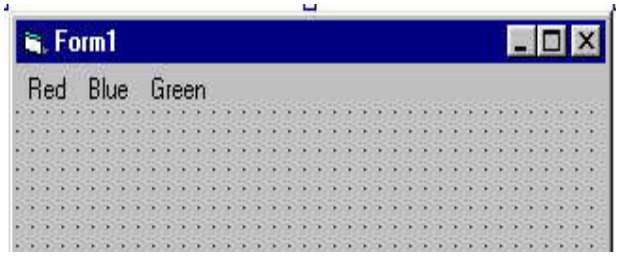
Suppose we want to create a menu for helping the user change the backcolor of the form to either red or blue or green. For this, we create a menu like this:

- Type 1st caption as 'Red', 1st name as 'mnured'. Click on 'Next' button.
- Type 2nd caption as 'Blue', 2nd name as 'mnublu'. Click on 'Next' button.
- Type 3rd caption as 'Green', 3rd name as 'mnugre'.

We should be seeing the menu editor, along with the menus as follows:



4. Click on the 'OK' button. The menu editor closes, and the form reappears, giving an appearance as shown below. Observe the menu that appears just below the title bar of the form:



Coding for Menus

The user interface for the menus is ready. Now we will code them to carry out the desired tasks when the user clicks on it. Click on a menu in the design window to open the code window. Now code for the menus click event:

```
Code   Listing   menu.1
PRIVATE  SUB  mnured_CLICK  (  )
form1.BACKCOLOR   =   VBRED
END  SUB
```

Notes

```
PRIVATE SUB mnublu_CLICK ( )
form1.BACKCOLOR = VBBLUE
END SUB

PRIVATE SUB mnugre_CLICK ( )
form1.BACKCOLOR = VBGREEN
END SUB
```

Coding for the menus is the same as coding for command buttons — coding for the 'click' event. Now run the project and test the menus by clicking on them. We will see the backcolor changing as we click on the menus.

Self Assessment

Fill in the blanks:

1. ..... tool helps us to effortlessly create a menu interface for our applications, that too in no time.
2. Index is required if we are creating a menu .....
3. The 'Enabled' and 'Visible' checkboxes are '.....' by default, which means that the menu will be both visible and enabled at the time of creation.
4. Coding for the menus is the same as coding for ..... buttons.
5. 'Caption' and 'Name' mean the same as we know; 'Caption' is the visible portion, 'Name' is for ..... of the application.
6. The '.....' option gives us a drop-down list for assigning shortcut keys to our menus.

10.2 Menu defined as Resource Data

"A resource is any non-executable data that is logically deployed with an application."

The easiest way to manage resource files in your project is to selecting the resources tab in the project properties. You can bring this up by double-clicking My Project in Solution Explorer or your project Properties under the Project menu item.

The resource types that are supported in the Resource Editor are:

- Strings
- Images (PNG, BMP, GIF, JPEG, and TIFF are supported!)
- Icons
- Audio
- Files
- Other

Using resource files adds another advantage: better globalization. (Microsoft has clearly heard the jingling clink of rupees, lira, yen, and krona in .NET.) Resources are normally included into your main assembly, but .NET also lets you package resources into satellite assemblies. This lets you accomplish better globalization because you can include just those satellite assemblies

Notes

when they're needed. Microsoft has given each language dialect a code. For example, the American dialect of English is indicated by the string "en-US", and the Swiss dialect of French is indicated by "fr-CH". These codes identify the satellite assemblies that contain culture specific resource files, for example: "en-AU.resx" for Australian English. When an application runs, Windows will automatically use the resources contained in the satellite assembly with the culture determined from Windows settings.

Since resources are a property of the solution in VB.NET, you access them just like other properties: by name, using the My.Resourcesobject. To make this more clear, let's build an application to display icons for Aristotle's four elements: Air, Earth, Fire and Water.

Step 1 is to add the icons. Select the Resources tab from your project Properties. You can add icons by either choosing Add Existing File... from the Add Resources drop down menu, or just drag and drop from a Windows Explorer window.

After a resource has been added, the new code looks like this:

```
Private Sub RadioButton1_CheckedChanged( ...
Handles MyBase.Load
Button1.Image = My.Resources.EARTH.ToBitmap
Button1.Text = "Earth"
End Sub
```

Although I do my best to keep these articles at a level where VB.NET Express can always be used, there is another way to use resources that isn't supported in Express. But if you're using Visual Studio, you can embed them directly in your project assembly. These steps will add an image directly into to your project.

- Right-click the project in the Solution Explorer, click Add, then click Add Existing Item.
- Browse to your image file and click Open.
- Display the properties for the image that was just added.
- Set the Build Action property to Embedded Resource.

You can then use the bitmap directly in code like this (the bitmap was the third one - index number 2 - in the assembly).

```
Dim res() As String = GetType(Form1).Assembly.GetManifestResourceNames()
PictureBox1.Image = New System.Drawing.Bitmap( _
   GetType(Form1).Assembly.GetManifestResourceStream(res(2)))
```

Although these resources are embedded as binary data directly in the main assembly (or in satellite assembly files) when you Build your project, in Visual Studio, they're referenced by an XML-based file format that use the extension .resx. For example, here's a snippet from the .resx file we just created:

```
<assembly alias="System.Windows.Forms" name="System.Windows.Forms,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <data name="AIR"
     type="System.Resources.ResXFileRef,
     System.Windows.Forms">
         <value>..\Resources\CLOUD.ICO;System.Drawing.Icon,
```

Notes

```
        System.Drawing, Version=2.0.0.0,
       Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a</value>
   </data>
```

You can write directly to these files at run time using the ResXResourceWriter object in the System.Resources namespace, even adding the actual binary information in a value XML element.

```
<value>
    AAEAAAD/////AQAAAAAAAAAMAgAAADtTeX…
</value>
```

ResXResourceReader, in turn, will read .resx files.

Because they're just text XML files, a .resx file can't be used directly by a .NET Framework application. It has to be converted to a binary ".resources" file adding it to your application. This job is accomplished by a utility program named Resgen.exe. You might want to do this to create the satellite assemblies for the globalization mentioned earlier. It can also convert binary .resources files back into XML .resx files but only string resources get converted correctly.

And DOS will never die! You have to run resgen.exe from a Command Prompt.

VB6 supports

A string table editor

("Edit String Tables...")

Custom cursors - "CUR" files

("Add Cursor...")

Custom icons - "ICO" files

("Add Icon...")

Custom bitmaps - "BMP" files

("Add Bitmap...")

Programmer defined resources

("Add Custom Resource...")

VB 6 provides a simple editor for strings but you have to have a file created in another tool for all of the other choices.

Example: You could create a BMP file using the simple Windows Paint program.

Each resource in the resource file is identified to VB 6 by an **Id** and a name in the Resource Editor. To make a resource available to your program, you add them in the Resource Editor then use the Id and the resource "Type" to point to them in your program. Let's add four icons to the resource file and use them in the program. When you add a resource, the actual file itself is copied into your project. Visual Studio 6 provides a whole collection of icons in the folder ...

C:\Program Files\Microsoft Visual Studio\Common\Graphics\Icons

To go with tradition, we'll select the Greek philosopher Aristotle's four "elements" - Earth, Water, Air, and Fire - from the Elements subdirectory. When you add them, the Id is assigned by Visual Studio (101, 102, 103, and 104) automatically.

Notes

Did u know? To use the icons in a program, we use a VB 6 "Load Resource" function.

There are several of "Load Resource" functions to choose from:

- **LoadResPicture(index, format)** for bitmaps, icons, and cursors

Use the VB predefined constants **vbResBitmap** for bitmaps, **vbResIcon** for icons, and **vbResCursor** for cursors for the "format" parameter. This function returns a picture that you can use directly. **LoadResData** (explained below) returns a string containing the actual bits in the file. We'll see how to use that after we demonstrate icons.

- **LoadResString(index)** for strings
- **LoadResData(index, format)** for anything up to 64K

Caution This function returns a string with the actual bits in the resource.

These are the values that can be used for format parameter here:

1. Cursor resource
2. Bitmap resource
3. Icon resource
4. Menu resource
5. Dialog box
6. String resource
7. Font directory resource
8. Font resource
9. Accelerator table
10. User-defined resource
11. Group cursor
12. Group icon

Since we have four icons in our About VB.RES resource file, let's use **LoadResPicture(index, format)** to assign these to the Picture property of a CommandButton in VB 6.

I created an application with four **OptionButton** components labeled Earth, Water, Air and Fire and four Click events - one for each option. Then I added a **CommandButton** and changed the Style property to "1 – Graphical". This is necessary to be able to add a custom icon to the CommandButton. The code for each OptionButton (and the Form Load event — to initialize it) looks like this (with the Id and Caption changed accordingly for the other OptionButton Click events):

```
Private Sub Option1_Click()
    Command1.Picture = _
        LoadResPicture(101, vbResIcon)
    Command1.Caption = _
       "Earth"
End Sub
```

Notes

Task Illustrate the use of LoadResData function.

Self Assessment

Fill in the blanks:

7. Each resource in the resource file is identified to VB 6 by an **Id** and a name in the .....
8. Use the VB predefined constants vbResBitmap for bitmaps, vbResIcon for icons, and vbResCursor for cursors for the "....." parameter.
9. To make a resource available to your program, you add them in the Resource Editor then use the Id and the resource "....." to point to them in your program.
10. When you add a resource, the actual file itself is ..... into your project.

10.3 Creating a Menu using the Borland Resource Workshop

Traditional Windows development suggests that you add version information to your compiled .EXE files. Microsoft provides two tools with Microsoft Visual C++ that allow Visual C++ programmers to add version information to compiled applications.

- The Microsoft Resource Compiler (RC.EXE).
- The Microsoft App Studio, a resource editor.

Unfortunately, at this time, Visual Basic programmers cannot add version information to Visual Basic generated .EXE files. Visual Basic does not have the ability to add resource information to its .EXE files. Nor can you use either the Microsoft Resource Compiler or the App Studio to add resource information to .EXE files generated by Visual Basic.

Notes There are however other third party Resource Compilers/Editors that will work with Visual Basic compiled .EXE files. One such tool is the "Resource Workshop" by Borland. You can contact Borland at 1-800-336-6464x8708.

Below is a step by step example illustrating how to add version information to a Visual Basic compiled EXE using Borland's "Resource Workshop."

Example: Example of Adding Version Information to a VB Application

1. Start Resource Workshop.
2. From the Resource Menu, choose New (ALT+R N).
3. Select VERSIONINFO as the resource type, and click OK.
4. Resource Workshop displays a default script for version information. Delete this, and type in something similar to the following:

```
1 VERSIONINFO LOADONCALL MOVEABLE FILEVERSION 1, 0, 0,
5 PRODUCTVERSION 1, 0, 0, 10 FILEOS VOS__WINDOWS16 FILETYPE
```

Notes

```
VFT_APP BEGIN
 BLOCK "StringFileInfo"
   BEGIN
       BLOCK "040904E4"
      BEGIN
            VALUE "CompanyName", "Some Company\000"
            VALUE "FileDescription", "What it is\000"
            VALUE "FileVersion", "03.00.0005\000"
            VALUE "InternalName", "XYZ.EXE\000"
            VALUE "LegalCopyright", "Copyright ) abcdefg"
            VALUE "LegalTrademarks", "Whatever you want\000"
            VALUE "ProductName", "asdfg\000"
            VALUE "ProductVersion", "see above"
            VALUE "Comments", "Some comments"
      END
   END
END
```

5. Double-click the control box to close the current window. Click Yes when Resource Workshop asks you if you wish to save changes.
6. Your Application now contains version information. Choose Exit from the File menu to exit Resource Workshop (ALT+F X). Click Yes when Resource Workshop asks you to verify that you want to update your EXE.

Self Assessment

Fill in the blank:

11. Visual Basic does not have the ability to add resource information to its ..... files.

10.4 Complex Menu

Now that you have an overview of how to use the Menu Editor to make a menu, the following example creates a menu system for a simple text editor.

The Amazing Text Editor, the code for which can do the following tasks:

- Create a new file
- Open an existing file
- Save a file
- Reverse the editor's font and background color setting
- Provide copyright notification
- Exit the program
- Undo the preceding action

Notes

- Cut, copy, and paste text
- Select all text

Before you start coding, take some time to review the program's features to make a properly designed and categorized menu system. As mentioned earlier in the section "Understanding the Windows Standard Menus," most menu bars begin with a File menu and are followed by an **Edit** menu.

The following is a viable menu categorization for the feature set of the Amazing Text Editor.

File Menu	Edit Menu
New	Undo
Open	Cut
Save	Copy
Settings	Paste
About	Select All
Exit	

Now that you have a categorized menu system, you can implement it in the Menu Editor. Table 10.1 shows the menu hierarchy and Name and Caption properties, as well as accelerator and shortcut keys for each menu object.

10.4.1 Standard Functionality of Shortcut Keys

The shortcut keys used in Table 10.1 adhere to the established convention that Windows programmers use for menu items with the demonstrated functionality.

Table 10.1: Menu Objects for the Amazing Text Editor Application

Name	Caption	Level	Shortcut
mnuFile	&File	0	None
itmNew	&New	1	None
itmOpen	&Open	1	None
itmSave	&Save	1	None
sepOne	- (a hyphen)	1	None
itmSettings	Se&ttings	1	None
itmBlackOnWhite	Black On White	2	None
itmWhiteOnBlack	White On Black	2	None
itmAbout	&About	1	None
sepTwo	-	1	None
itmExit	E&xit	1	Ctrl+X
mnuEdit	&Edit	0	None
itmUndo	&Undo	1	Ctrl+Z
sepThree	-	1	None
itmCut	Cu&t	1	Ctrl+X
itmCopy	&Copy	1	Ctrl+C
itmPaste	&Paste	1	Ctrl+V
sepFour	-	1	None
itmSelectAll	Select &All	1	Ctrl+A

Notes

By default, Flyout Menus are positioned with the top-left corner being 2 pixels to the right and 2 pixels above the top-right corner of the image defined with the same tag name as the menu. Variables can be set to move the menu to the left of the image, or to change the number of pixels away from the image. For most uses, these variables offer menu position options which are quite flexible.

However, there may be occasions when menus must be positioned taking more images into account on the page.

Did u know? The Flyout Menus allow you set up a series of positioning rules for more complex positioning behavior.

There is quite a bit of complexity when it comes to using this positioning code, so it may take several tries before the menus appear correctly positioned.

Notes Due to the way its object model and styles are implemented, Internet Explorer for the Macintosh is very slow at using the complex menu positioning. The explanation for why this browser is different is in the annotated source.

Complex Positioning Example

As an example, the menus below are set so that they will never cover up the text in the logo image (overwriting the box is fine), and attempt to not go below the picture image. In either case, they will try to get as close as possible to having the menu centered vertically with its arrow image. The exception to these positioning rules is that the menu will always attempt to not appear off the page, so it will move to prevent that from happening. Keeping the menu on the page takes precedence over all other rules, and cannot be overridden. If the menu is too tall to fit at all on the page, it will be positioned so as much of the top-left part is visible as possible.

This is the source of the menu code which includes the positioning rules:

```
var newDefs = new Object;
newDefs.overimg = "redarrow.gif";
newDefs.useclass = "menutext";
newDefs.position = "IMG|m=m;picture|b>b;logo|b<t";
flyDefs (newDefs);
makeLayer ("arrow1", "",
       "This is menu 1", "It should be positioned", "so the top edge is right",
       "under the logo."
);
makeLayer ("arrow2", "",
       "This is menu 2", "It should be centered", "relative to the arrow"
);
makeLayer ("arrow3", "",
```

Notes

```
        "This is menu 3", "It is quite a long menu", "so it should also be",
        "positioned so the top", "edge is right under the", "logo, but the
bottom",
        "edge may come close to", "the bottom of the picture,", "or may even
go below it.",
        "If you size and scroll the", "window to try to force the", "bottom
of the menu off below",
        "the bottom of the screen,", "it will reposition itself", "by
obscuring the logo."
);
makeLayer ("arrow4", "",
        "This is menu 4", "It should be long enough", "to be aligned with
the",
        "bottom of the picture, but", "probably won't reach the", "logo."
);
```

This looks like the menu definitions in the Customizing Flyout Menus section, except for the line which defines a new position default. Let's first look at the logic of what the line is doing, and then describe the syntax.

1. The Flyouts will use the default positioning rules to put the top-left corner of the menu near the top-right corner of the arrow image.
2. The IMG|m=m rule tells the Flyout code to compute the vertical middle of both the arrow image and the menu, and position the menu so they are the same.
3. The picture|b>b rule says to compute the bottom of the image named *picture* and of the menus, and make sure the bottom of the picture is positioned lower than (has a larger number, representing the Y position) the bottom of the menu. If not, then move the menu up so they align.
4. The logo|b-6<t rule will compute 6 pixels above the bottom of the image named *logo* and the top of the menu, and make sure 6 pixels above the bottom of the logo is positioned higher than (has a smaller number, again representing the Y position) the top of the menu. If not, move the menu down so they align.
5. Make sure the menu is on the screen. First check horizontally, moving first to the left if it's off the right of the window, then to the right to make sure the left of the menu is visible. Do the same vertically, first checking the bottom, then the top.

The way these rules are ordered means that the preference is for the menus to be aligned with the middle of the arrow, the next preference is that they not go below the picture, but the greatest preference is they not cover up the logo.

Task Discuss how Flyout Menus are positioned?

Self Assessment

Fill in the blanks:

12. Before you start coding, take some time to review the program's features to make a properly designed and categorized .....

Notes

13. The ..... Menus allow you set up a series of positioning rules for more complex positioning behavior.

10.5 Creating a Menu as a Program Operates

This code shows you how to add a menu to another program. The only thing is that nothing will happen when you click on the items. To make something happen when you click on an item you have to subclass the menu (I'd help with that but I don't have any subclassing controls, or at least not right now). Put this in your *.bas file:

```
Public Declare Function AppendMenu Lib "user32" Alias "AppendMenuA" (ByVal hMenu As Long, ByVal wFlags As Long, ByVal wIDNewItem As Long, ByVal lpNewItem As Any) As Long

Public Declare Function CreatePopupMenu Lib "user32" () As Long

Public Declare Function DrawMenuBar Lib "user32" (ByVal hwnd As Long) As Long

Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Public Declare Function GetMenu Lib "user32" (ByVal hwnd As Long) As Long

Public Declare Function GetMenuItemID Lib "user32" (ByVal hMenu As Long, ByVal nPos As Long) As Long

Public Declare Function GetMenuItemCount Lib "user32" (ByVal hMenu As Long) As Long

Public Declare Function GetMenuString Lib "user32" Alias "GetMenuStringA" (ByVal hMenu As Long, ByVal wIDItem As Long, ByVal lpString As String, ByVal nMaxCount As Long, ByVal wFlag As Long) As Long

Public Declare Function GetSubMenu Lib "user32" (ByVal hMenu As Long, ByVal nPos As Long) As Long

Public Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long


Public Const MF_ENABLED = &H0&

Public Const MF_POPUP = &H10&

Public Const MF_STRING = &H0&

Public Const WM_NCPAINT = &H85

Then put something like this in a button:

Dim newMenu As Long

newMenu = CreatePopupMenu

Call AppendMenu(newMenu, MF_ENABLED Or MF_STRING, 0, "Item One")

Call AppendMenu(newMenu, MF_ENABLED Or MF_STRING, 1, "Item Two")

Call AppendMenu(newMenu, MF_ENABLED Or MF_STRING, 2, "Item Three")

Call AppendMenu(newMenu, MF_ENABLED Or MF_STRING, 3, "Item Four")

Call AppendMenu(newMenu, MF_ENABLED Or MF_STRING, 4, "Item Five")


' Find the notepad application window
```

Notes

```
Dim notepad As Long
notepad = FindWindow("notepad", vbNullString)

' Add our menu to the window we found above
Dim notepadMenu As Long
notepadMenu = GetMenu(notepad)
Call AppendMenu(notepadMenu, MF_POPUP, newMenu, "Item List")

' Ensure that the user sees the new menu immediately
Call SendMessage(notepad, WM_NCPAINT, 0&, 0&)
```

Self Assessment

Fill in the blank:

14. To make something happen when you click on an item you have to ..... the menu.

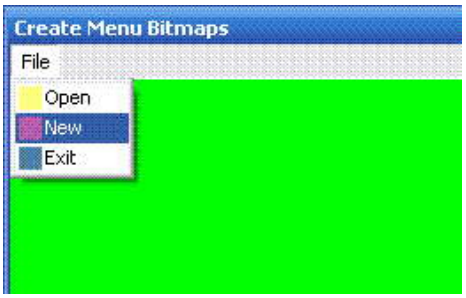
10.6 Creating Menu Containing Bitmaps and the System Menu

10.6.1 Creating Menu Containing Bitmaps

Create Menu Bitmaps With VB6 API

To create Menu Bitmaps With VB6 API, open the 'create menu bitmap' folder.

From file, click new to create menu.




```
' VB6 API Source Code Option Explicit
Private Const Color_Green = vbGreen
Private Declare Function GetMenu Lib "user32" (ByVal hwnd As Long) As
Long
Private Declare Function GetSubMenu Lib "user32" (ByVal hMenu As Long,
ByVal nPos As Long) As Long
Private Declare Function SetMenuItemBitmaps Lib "user32" (ByVal hMenu As
Long, _
ByVal nPosition As Long, ByVal wFlags As Long, ByVal hBitmapUnchecked As
Long, _
```

Notes

```
ByVal hBitmapChecked As Long) As Long
Private Declare Function GetMenuItemID Lib "user32" (ByVal hMenu As Long,
ByVal nPos As Long) As Long
Private Sub Form_Load()
Dim hMenu As Variant
Dim hSubMenu As Variant
Dim MenuID As Variant
Dim lX As Variant
VB6_API_frmCreateMenuBitmaps.BorderStyle = vbSizableToolWindow
VB6_API_frmCreateMenuBitmaps.Caption = " Create Menu Bitmaps "
VB6_API_frmCreateMenuBitmaps.BackColor = Color_Green
hMenu = GetMenu(VB6_API_frmCreateMenuBitmaps.hwnd)
hSubMenu = GetSubMenu(hMenu, 0)
MenuID = GetMenuItemID(hSubMenu, 0)
lX = SetMenuItemBitmaps(hMenu, MenuID, &H4, _
ImgLst.ListImages(1).Picture, ImgLst.ListImages(1).Picture)
MenuID = GetMenuItemID(hSubMenu, 1)
lX = SetMenuItemBitmaps(hMenu, MenuID, &H4, _
ImgLst.ListImages(2).Picture, ImgLst.ListImages(2).Picture)
MenuID = GetMenuItemID(hSubMenu, 2)
lX = SetMenuItemBitmaps(hMenu, MenuID, &H4, _
ImgLst.ListImages(3).Picture, ImgLst.ListImages(3).Picture)
End Sub
Private Sub mnuExit_Click()
Dim MsgResult As VbMsgBoxResult
MsgResult = MsgBox(" You Choose Menu " & mnuExit.Caption & " Do you want to
exit form ? ", vbQuestion + vbYesNo, " Create Menu Bitmaps With VB6 API ")
If MsgResult = vbYes Then
End
End If
End Sub
Private Sub mnuNew_Click()
MsgBox " You Choose Menu " & mnuNew.Caption, vbOKOnly, " Create Menu Bitmaps
With VB6 API "
End Sub
Private Sub mnuOpen_Click()
MsgBox " You Choose Menu " & mnuOpen.Caption, vbOKOnly, " Create Menu
Bitmaps With VB6 API "
End Sub
```

Notes

10.6.2 The System Menu

This example adds an About... item to the forms System Menu (shown by clicking the forms icon, or right clicking on its button on the taskbar)

Caution This uses subclassing, so DO NOT use the Stop button on the VB toolbar, or attempt to debug the WindowProc procedure (unless you like VB crashing!).

First, add the following code to a form

```
'// form_load event. Catch all those messages!
Private Sub Form_Load()
Dim lhSysMenu As Long, lRet As Long
On Error Resume Next
'// add about menu
lhSysMenu = GetSystemMenu(hWnd, 0&)
lRet = AppendMenu(lhSysMenu, MF_SEPARATOR, 0&, vbNullString)
lRet = AppendMenu(lhSysMenu, MF_STRING, IDM_ABOUT, "About...")
Show
'// saves the previous window message handler. Always restore this value
'// Address Of command sends the address of the WindowProc procedure
'// to windows
ProcOld = SetWindowLong(hWnd, GWL_WNDPROC, Address Of WindowProc)
End Sub
'// form_queryunload event. Return control to windows/vb
Private Sub Form_Unload(Cancel As Integer)
'// give message processing control back to VB
'// if you don't do this you WILL crash!!!
Call SetWindowLong(hWnd, GWL_WNDPROC, ProcOld)
End Sub
```

Then, add the code below to a module

```
'// variable that stores the previous message handler
Public ProcOld As Long
'// Windows API Call for catching messages
Public Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA"
(ByVal hWnd
As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
'// Windows API call for calling window procedures
Public Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA"
(ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal
wParam As Long,
```

Notes

```
ByVal lParam As Long) As Long
'// menu windows api
Declare Function AppendMenu Lib "user32" Alias "AppendMenuA" (ByVal hMenu
As Long,
ByVal wFlags As Long, ByVal wIDNewItem As Long, ByVal lpNewItem As String)
As Long
Declare Function GetSystemMenu Lib "user32" (ByVal hWnd As Long, ByVal
bRevert As Long)
As Long
'// windows api constants
Public Const WM_SYSCOMMAND = &H112
Public Const MF_SEPARATOR = &H800&
Public Const MF_STRING = &H0&
Public Const GWL_WNDPROC = (-4)
Public Const IDM_ABOUT As Long = 1010
Public Function WindowProc(ByVal hWnd As Long, ByVal iMsg As Long, _
ByVal wParam As Long, ByVal lParam As Long) As Long
     '// ——WARNING——
     '// do not attempt to debug this procedure!!
     '// ——WARNING——

     '// this is our implementation of the message handling routine
     '// determine which message was received
     Select Case iMsg
     Case WM_SYSCOMMAND

         If wParam = IDM_ABOUT Then
              MsgBox "VB Web Append to System Menu Example", vbInformation,
"About"
             Exit Function
        End If
     End Select

     '// pass all messages on to VB and then return the value to windows

      WindowProc = CallWindowProc(ProcOld, hWnd, iMsg, wParam, lParam)
End Function
```

Add New Menu To System Menu

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
```

Notes

```
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;
.........
            #region API Declarations
            // return the handle for the system menu
            [DllImport ("user32.dll")]
            public static extern int GetSystemMenu(int hwnd, bool bRevert);
            // append a menu item to the menu
            [DllImport ("user32.dll", EntryPoint="AppendMenuA")]
            public static extern long AppendMenu(int hMenu, int
            wFlags, int wIDNewItem, string lpNewItem);
            // remove a menu item from the menu
            [DllImport ("user32.dll")]
            public static extern long RemoveMenu(int hMenu, int
            nPosition, int wFlags);
            #endregion // API Declarations
            // constants
            private const int MF_BYPOSITION = 1024;
            private const int MF_SEPERATOR = 2048;
            private const int MF_REMOVE = 4096;
            private const int WM_SYSCOMMAND = 274;
            #region System Menu API
            /// <summary>
            /// Returns the handle for the System Menu with the
                 associate form
            /// </summary>
            /// <param name="frmHandle">The handle to the form to retrieve
the menu handle</param>
            public int GetSysMenuHandle(int frmHandle)
            {
                   return GetSystemMenu(frmHandle, false);
            }
            /// <summary>
            /// Removes a system menu
            /// </summary>
            /// <param name="mnuHandle">Then handle to the system menu.
Use GetSysMenuHandle</param>
            /// <param name="mnuPosition">Zero based position of the
menu item to delete</param>
            /// <returns>returns nonzero on success, returns 0 on fail</
returns>
            public long RemoveSysMenu(int mnuHandle, int mnuPosition)
```

Notes

```
{
        return RemoveMenu(mnuHandle, mnuPosition, MF_REMOVE);
}
/// <summary>
/// Appends a menu item to the end of the system menu list
/// </summary>
/// <param name="mnuHandle">Handle to the system menu to
    append to</param>
/// <param name="MenuID">A unique ID sent by the calling
    function to track events for the menu</param>
/// <param name="mnuText">The display text of the menu</
    param>
/// <returns>Returns nonzero on success, 0 on fail.</returns>
public long AppendSysMenu(int mnuHandle, int MenuID,
string mnuText)
{
        return AppendMenu(mnuHandle, 0, MenuID, mnuText);
}
/// <summary>
/// Appends a separator bar at the end of the system menu
/// </summary>
/// <param name="mnuHandle">Handle to the system menu</param>
/// <returns>Returns nonzero on success, 0 on fail</returns>
public long AppendSeparator(int mnuHandle)
{
        return AppendMenu(mnuHandle, MF_SEPARATOR, 0, null);
}
// THIS IS IMPORTANT. The WndProc function has to be
   overridden so
// we can capture the events from a menu item being selected.
// In the future, we could add OwnerDraw features here,
   and process
// that here, too
/// <param name="messg">The messages sent to WndProc</param>
protected override void WndProc(ref Message messg)
{
        int i = 1; // for EXAMPLE purposes only
        // see if this is a menu message to monitor
        if (messg.Msg == WM_SYSCOMMAND)
        {
                // decide how to handle the command
                switch (messg.WParam.ToInt32())
                {
```

Notes

```
                                case x:
                                // TODO: this is for example only, replace
                                                                    x with
                                // whatever values you gave for the
                                                                   MenuID
                                // then process the menu event here by
                                                                  calling
                                // whatever function necessary
                            }
                        }
                        // return the message so other wndprocs can process
them
                        base.WndProc(ref messg);
                }
                #endregion // System Menu API
```

Self Assessment

Fill in the blank:

15. System Menu uses subclassing, so DO NOT use the Stop button on the VB toolbar, or attempt to debug the ..... procedure.

10.7 Summary

- For getting started with menus, we use Menu Editor Tool that helps us to effortlessly create a menu interface for our applications, that too in no time.
- Coding for the menus is the same as coding for command buttons - coding for the 'click' event.
- VB 6 provides a simple editor for strings but you have to have a file created in another tool for all of the other choices.
- To use the icons in a program, we use a VB 6 "Load Resource" function.
- Variables can be set to move the menu to the left of the image, or to change the number of pixels away from the image.
- Microsoft provides two tools with Microsoft Visual C++ that allow Visual C++ programmers to add version information to compiled applications i.e., the Microsoft Resource Compiler (RC.EXE) and the Microsoft App Studio, a resource editor.
- The Flyout Menus allow you set up a series of positioning rules for more complex positioning behavior.
- The Flyouts will use the default positioning rules to put the top-left corner of the menu near the top-right corner of the arrow image.

10.8 Keywords

Flyout Menus: The Flyout Menus allow you set up a series of positioning rules for more complex positioning behavior.

Notes

10.9 Review Questions

1. Illustrate the steps for creating menus.
2. Explain how to use the menu editor tool.
3. Write the code for the menus click event.
4. Enlighten the various types of "Load Resource" functions.
5. What are the VB predefined constants used for bitmaps, icons, and cursors? Illustrate.
6. How to Create a menu using the Borland Resource Workshop? Illustrate with example.
7. What are the two tools used with Microsoft Visual C++ that allow Visual C++ programmers to add version information to compiled applications? Discuss.
8. Illustrate the concept of complex menu with example.
9. Enlighten the steps used to create a menu containing Bitmaps.
10. Write a code for adding new Menu to System Menu.

Answers: Self Assessment

1.	Menu Editor	2.	array
3.	checked	4.	command
5.	coding	6.	Shortcut
7.	Resource Editor	8.	format
9.	Type	10.	copied
11.	.EXE	12.	menu system
13.	Flyout	14.	subclass
15.	WindowProc		

10.10 Further Readings

Books

Brent E. Rector, *Win32 Programming,* Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

www.reddit.com/.../programming/.../toplevel_windows_and_menus

Notes

Unit 11: Dialog Boxes (I)

CONTENTS

Objectives

Introduction

11.1 What is a Dialog Box?

11.2 How a Dialog Box Works?

11.3 Designing a Dialog Box

11.3.1 Create the Dialog Box

11.3.2 Adding the Controls

11.4 Using a Dialog Box

11.5 Summary

11.6 Keywords

11.7 Review Question

11.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of dialog boxes
- Discuss the working of dialog box
- Recognize designing a dialog box and using a dialog box

Introduction

All the GUI applications keep prompting and conversing to the users, by means of dialog boxes. Dialog boxes have been the only interface by which an application can converse to its users, and it will be till the Speech Oriented Software become practicable. This unit covers the concept of designing dialog boxes and using dialog boxes.

11.1 What is a Dialog Box?

A dialog box is defined as a rectangular window whose major role is to hold other Windows controls. For this cause, a dialog box is pointed to as a container. It is the chief interface of user communication with the computer. By itself, a dialog box means nothing. The controls it holds achieve the role of dialog among the user and the machine.

A dialog box has the following traits:

- It is equipped with the system Close button . As the only system button, this button permits the user to release the dialog and overlook 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.

Notes

- It is typically modal. The user is typically not permitted to carry on any other operation until the dialog box is dismissed.

Self Assessment

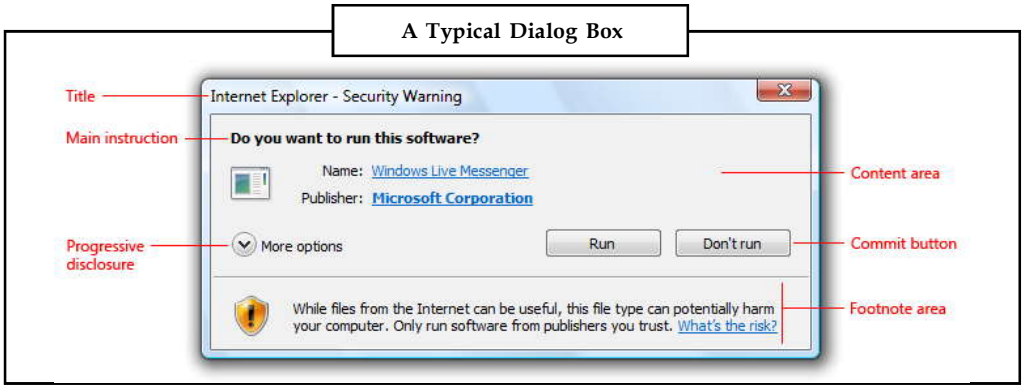
Fill in the blanks:

1. A ..... is defined as a rectangular window whose major role is to hold other Windows controls.
2. A dialog box is the chief interface of user ..... with the computer.

11.2 How a Dialog Box Works?

A *dialog box* is a secondary window that allows users to perform a command, asks users a question, or provides users with information or progress feedback.

A Typical Dialog Box



Dialog boxes consist of a title bar (to identify the command, feature, or program where a dialog box came from), an optional main instruction (to explain the user's objective with the dialog box), various controls in the content area (to present options), and commit buttons (to indicate how the user wants to commit to the task).

Dialog boxes have two fundamental types:

1. ***Modal dialog boxes*** require users to complete and close before continuing with the owner window. These dialog boxes are best used for critical or infrequent, one-off tasks that require completion before continuing.
2. ***Modeless dialog boxes*** allow users to switch between the dialog box and the owner window as desired. These dialog boxes are best used for frequent, repetitive, ongoing tasks.

A *task dialog* is a dialog box implemented using the task dialog application programming interface (API). They consist of the following parts, which can be assembled in a variety of combinations:

- A **title bar** to identify the application or system feature where the dialog box came from.
- A **main instruction**, with an optional icon, to identify the user's objective with the dialog.
- A **content area** for descriptive information and controls.
- A **command area** for commit buttons, including a Cancel button, and optional More options and *Don't show this <item> again* controls.
- A **footnote area** for optional additional explanations and help, typically targeted at less experienced users.

Notes

A Typical Task Dialog

Did u know? Task dialogs are recommended whenever appropriate because they are easy to create and they achieve a consistent look.

Caution Task dialogs do require Windows Vista® or later, so they aren't suitable for earlier versions of Microsoft® Windows®.

A *task pane* is like a dialog box, except that it is presented within a window pane instead of a separate window. As a result, task panes have a more direct, contextual feel than dialog boxes. Property windows are a specialized type of dialog box used to view and change properties for an object, collection of objects, or a program. Additionally, property windows typically support several tasks, whereas dialog boxes typically support a single task or step in a task.

Notes Dialog boxes can have tabs, and if so they are called *tabbed dialog boxes*. Property windows are determined by their presentation of properties, not by the use of tabs.

Task What is *task dialog*? Illustrate.

Self Assessment

Fill in the blanks:

3. A *dialog box* is a ..... window that allows users to perform a command, asks users a question, or provides users with information or progress feedback.
4. A ..... is a dialog box implemented using the task dialog application programming interface (API).
5. A ..... is used for descriptive information and controls.
6. A ..... is used for optional additional explanations and help, typically targeted at less experienced users.

Notes

7. A ..... is used to identify the application or system feature where the dialog box came from.
8. A ..... is like a dialog box, except that it is presented within a window pane instead of a separate window.
9. ..... windows typically support several tasks, whereas dialog boxes typically support a single task or step in a task.
10. Dialog boxes can have tabs, and if so they are called .....
11. Property windows are a specialized type of dialog box used to view and change ..... for an object, collection of objects, or a program.

11.3 Designing a Dialog Box

The Visual C++ environment offers a dialog resource editor for designing dialog boxes. This editor exhibits the **Controls** toolbar, which displays the obtainable controls (like radio buttons, check boxes, and pushbuttons). You choose controls from the **Controls** toolbar and place them on your dialog box.

Notes You can move and resize the controls directly by means of the mouse.

Caution You utilize the property page for every control to state its caption and ID.

Designing a dialog box follow the steps below.

1. Creating a new dialog box and editing its caption and ID.
2. Adding the controls and editing their captions and IDs.

11.3.1 Create the Dialog Box

Now you'll create a simple dialog box by beginning with the default dialog box that the development environment offers.

Let us consider a dialog box to be created named as Box1.

To create the Box1 dialog box

1. With your project open, click **Resource** from the **Insert** menu,
2. In the **Insert Resource** dialog box, choose **Dialog** from the list of resource types and click **OK**. ResourceView pane opens and the dialog editor window occurs, showing a default dialog box that includes two buttons labeled OK and Cancel. The **Controls** toolbar also occurs.
3. If the property page is not shown, right-click the dialog box, then click **Properties** on the menu. Click the pushpin on the property page to keep it open.
4. In the ID box, type IDD_Box1. This is not a predefined ID, so you can't choose it from the drop-down list.

Notes

5. In the caption box, modify the caption to Pen Widths. Observe that the title bar of the dialog box reflects the new caption.
6. Save your work.

11.3.2 Adding the Controls

To add controls to the dialog box:

1. From the **Controls** toolbar, add two edit box controls to the Box1 dialog box.
   - Choose the first edit box to exhibit its property page. Change its ID to IDC_THIN_Box1.
   - Choose the second edit box, and in the property page alter its ID to IDC_THICK_Box1
2. From the **Controls** toolbar, add two static text controls to enclose the descriptions for the two edit controls.
3. Choose the first text box to exhibit its property page. Modify the caption to Thin Pen Width:.
4. Choose the second text box, and in the property page change its caption to Thick Box1:.
5. From the **Controls** toolbar, add a third button to the two already present.
6. Choose the third button to exhibit its property page. Transform its ID to IDC_DEFAULT_Box1 and its caption to Default.

Did u know? The handler for this button will reset the thick and thin Box1 to their default widths.

Self Assessment

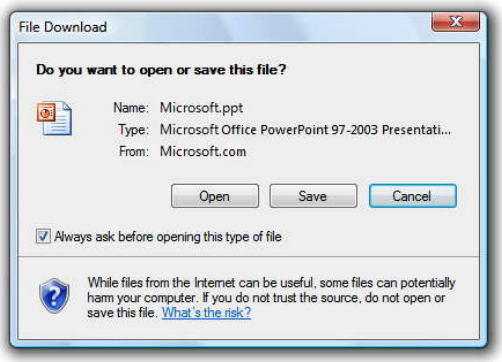
Fill in the blanks:

12. The Visual C++ environment offers a ..... editor for designing dialog boxes.
13. You utilize the ..... for every control to state its caption and ID.

11.4 Using a Dialog Box

Dialog boxes have several usage patterns:

Question dialogs (using buttons) Ask users a single question, and provide simple responses in horizontally arranged command buttons.



Notes

Example: Windows® Internet Explorer® asks if the user wants to open or save a file.

Type: Modal.

Main instruction: The question being asked (could be phrased as an instruction).

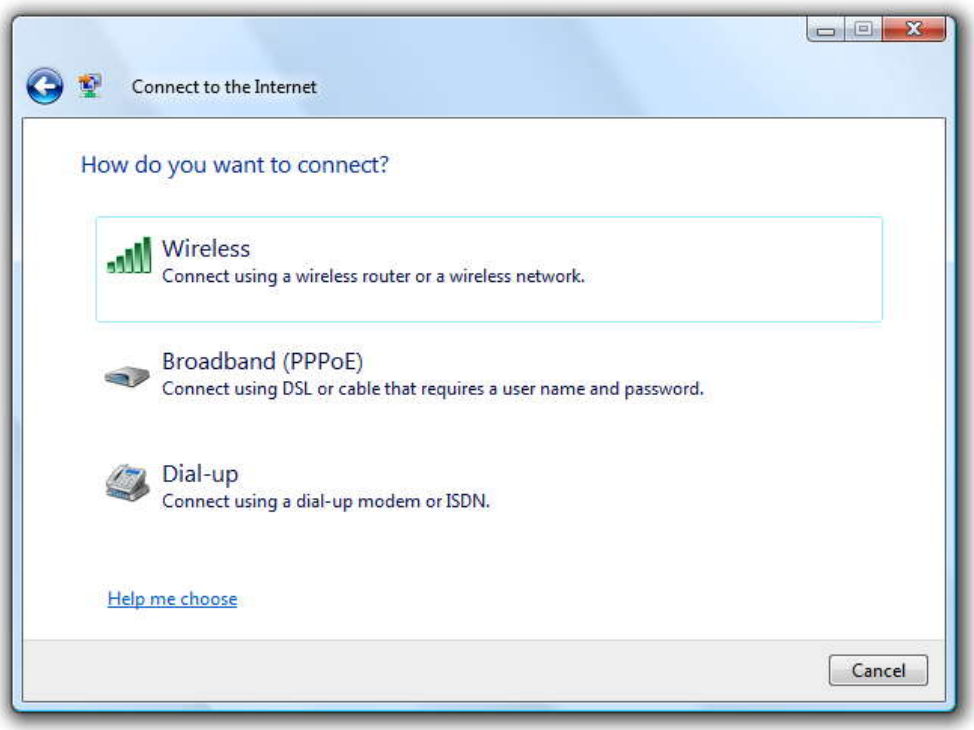
Icon: Program, feature, object, warning icon (if potential loss of data or system access), security warning, or none.

Commit buttons: One of the following sets of concise commands: Yes/No, Yes/No/Cancel, [Do it]/Cancel, [Do it]/[Don't do it], [Do it]/[Don't do it]/Cancel, where [Do it] and [Don't do it] are specific responses to the main instruction.

Other controls: There may be supplemental explanations to help users make informed decisions, a chevron control to show more information, and a *Don't show this <item> again* option if the question can be suppressed in the future.

Annoyance factor: High, if default response can be safely assumed, there really isn't a choice, or the differences among the choices aren't clear.

Question dialogs (using command links): Ask users a single question or to select a task to perform, and provide detailed responses in vertically arranged command links.



Example: Windows asks the user to install a device. Using command links instead of command buttons allows for more complete responses. In contrast to the version with command buttons, these dialogs may have several responses or responses that require more text to describe.

Type: Modal.

Main instruction: The question being asked (could be phrased as an instruction).

Notes

Icon: Program, feature, object, warning icon (if potential loss of data or system access), security warning, or none.

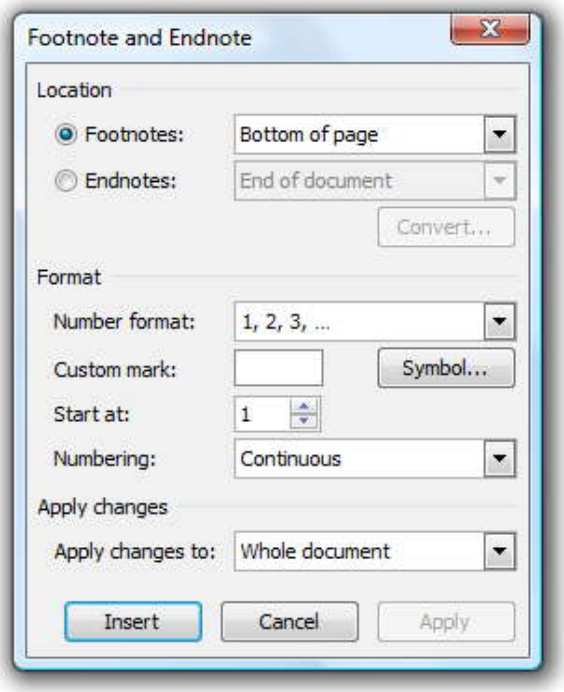
Command links: Two or more complete, specific responses to the main instruction.

Commit buttons: Cancel.

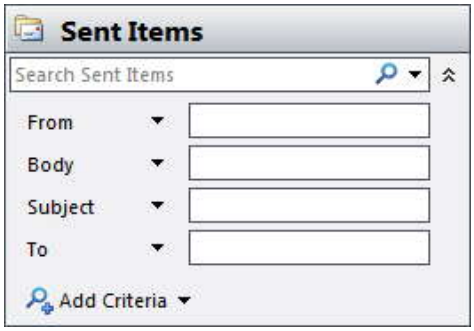
Other controls: There may be supplemental explanations to help users make informed decisions, and a chevron to show more information.

Annoyance factor: High, if default response can be safely assumed, there really isn't a choice, or the differences among the choices aren't clear.

Choice dialogs Presents users with a set of choices, usually to specify a command more completely. Unlike question dialogs, choice dialogs can ask multiple questions.



Example: Microsoft Word presents options to specify the Insert Break command in a modal dialog box. In this example, Word presents options to specify the Find and Replace command in a modeless dialog box.



Notes

Example: Microsoft Outlook® presents options to specify the Find command in a task pane. By not using a separate window, the command feels more direct and contextual.

Type: Modal, modeless, and task pane.

Main instruction: An optional imperative instruction that tells users what to do.

Icon: None.

Commit buttons: One of the following:

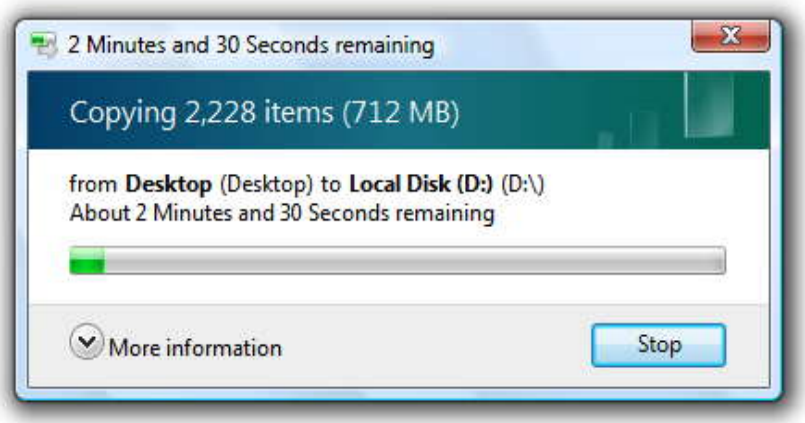
- ***Modal dialogs:*** OK/Cancel or [Do it]/Cancel, where [Do it] is a specific response to the main instruction.
- ***Modeless dialogs:*** Close button on dialog box and title bar.
- ***Task pane:*** Close button on title bar.

Other controls: There may be supplemental explanations to help users make choices, and a chevron to show infrequently used options.

Annoyance factor: Normally low, because user initiated and needs a response, but could be high if users rarely change default values.

Progress dialogs: Presents users with progress feedback during a lengthy operation (longer than five seconds), along with a command to cancel or stop the operation.

If the operation is a long-running task (over 30 seconds) and can be performed in the background, use a modeless progress dialog so that users can continue to use your program while waiting.



Example: A modeless progress dialog box is used provide feedback while users continue to use the program.

Type: Modal and modeless.

Main instruction: A gerund phrase briefly explaining the operation in progress, ending with an ellipsis. Example: Downloading...

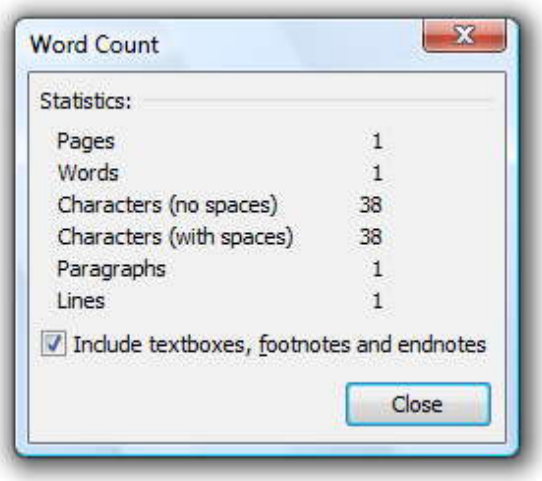
Icon: None (but may have an animation).

Commit buttons: Use Cancel if returns the environment to its previous state (leaving no side effect); otherwise, use Stop.

Notes

Annoyance factor: Low, if user needs to know when operation is complete, but high if unnecessarily modal or operation isn't significant.

Informational dialogs Display information requested by the user.



Example: Word uses a modal dialog box to display word count information.

Type: Modal.

Main instruction: A sentence that describes the information.

Icon: None.

Commit buttons: Close

Other controls: There may be a chevron to show more information.

Annoyance factor: Low, if information is relevant and requested by the user.

Self Assessment

Fill in the blanks:

14. Question dialogs (using buttons) ask users a single question, and provide simple responses in horizontally arranged ..... buttons.

15. ..... dialogs are used to display information requested by the user.

11.5 Summary

- All the GUI applications keep prompting and talking to the users, using dialog boxes.
- A dialog box is defined as a rectangular window whose major role is to hold other Windows controls.
- The controls that a dialog box holds achieve the role of dialog among the user and the machine.
- A *dialog box* is a secondary window that allows users to perform a command, asks users a question, or provides users with information or progress feedback.

Notes

- A *task dialog* is a dialog box implemented using the task dialog Application Programming interface (API).
- A *task pane* is like a dialog box, except that it is presented within a window pane instead of a separate window.
- Dialog boxes can have tabs, and if so they are called *tabbed dialog boxes*.
- Property windows typically support several tasks, whereas dialog boxes typically support a single task or step in a task.

11.6 Keywords

Caption: This is the name of the menu that will appear on the menu bar.

Dialog Box: A dialog box is defined as a rectangular window whose major role is to hold other Windows controls.

Task Dialog: A *task dialog* is a dialog box implemented using the task dialog Application Programming Interface (API).

Task Pane: A *task pane* is like a dialog box, except that it is presented within a window pane instead of a separate window.

11.7 Review Questions

1. What is a dialog box? Illustrate the characteristics of dialog box.
2. Show how a dialog box work.
3. Illustrate the process of implementing task dialog.
4. What is a task pane? Explain why task panes have a more direct, contextual feel than dialog boxes.
5. Illustrate the use of *tabbed dialog boxes.*
6. What are the steps needed in designing a Dialog Box? Discuss.
7. Illustrate the steps required in creating a dialog box with example.
8. Depict the several usage patterns defined by dialog box.
9. Discuss the various parts of task dialog, which can be assembled in a variety of combinations.
10. Illustrate the steps used in adding the controls to the dialog box.

Answers: Self Assessment

1.	dialog box	2.	communication
3.	secondary	4.	task dialog
5.	content area	6.	footnote area
7.	title bar	8.	task pane
9.	Property	10.	tabbed dialog boxes.
11.	properties	12.	dialog resource

13. property page
14. command
15. Informational

Notes

11.8 Further Readings

Books

Brent E. Rector, *Win32 Programming*, Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

www.functionx.com/win32/Lesson04.htm

Notes

Unit 12: Dialog Boxes (II)

CONTENTS

Objectives

Introduction

12.1 Exchanging Data with a Dialogue Box – Global Variable Method and Pointer Method

12.2 Problems in Global Variables

12.2.1 Why Global Variables should be avoided when Unnecessary?

12.2.2 Why the Convenience of Global Variables sometimes Outweighs the Potential Problems?

12.3 Exchanging Data with a Dialog Box – Pointer Method

12.4 Modal, Modeless and System Modal Dialog Boxes

12.4.1 Modal Dialog Box

12.4.2 Modeless Dialog Box

12.4.3 System Modal Dialog Box

12.5 Creating a Modeless Dialog Box

12.6 Summary

12.7 Keywords

12.8 Review Questions

12.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the concept of exchanging data with a dialog box using global variable method and pointer method
- Discuss problems in global variables
- Understand modal, modeless, and system modal dialog boxes
- Illustrate creating modeless dialog box

Introduction

In this unit you will understand the process of exchanging data with a dialog box by means of Global Variable Method and Pointer Method. Also you will recognize various types of dialog boxes such as modal dialog box, modeless dialog box, and system modal dialog box. Creating a modeless dialog box is also discussed in this unit.

Notes

12.1 Exchanging Data with a Dialogue Box – Global Variable Method and Pointer Method

This topic lists the DDX_OC functions used to exchange data between a property of an OLE control in a dialog box, form view, or control view object and a data member of the dialog box, form view, or control view object.

DDX_OC Functions

DDX_OCBool	Manages the transfer of **BOOL** data between a property of an OLE control and a **BOOL** data member.
DDX_OCBoolRO	Manages the transfer of **BOOL** data between a read-only property of an OLE control and a **BOOL** data member.
DDX_OCColor	Manages the transfer of **OLE_COLOR** data between a property of an OLE control and an **OLE_COLOR** data member.
DDX_OCColorRO	Manages the transfer of **OLE_COLOR** data between a read-only property of an OLE control and an **OLE_COLOR** data member.
DDX_OCFloat	Manages the transfer of **float** (or **double**) data between a property of an OLE control and a **float** (or **double**) data member.
DDX_OCFloatRO	Manages the transfer of **float** (or **double**) data between a read-only property of an OLE control and a **float** (or **double**) data member.
DDX_OCInt	Manages the transfer of int (or **long**) data between a property of an OLE control and an int (or **long**) data member.
DDX_OCIntRO	Manages the transfer of int (or **long**) data between a read-only property of an OLE control and an int (or **long**) data member.
DDX_OCShort	Manages the transfer of **short** data between a property of an OLE control and a **short** data member.
DDX_OCShortRO	Manages the transfer of **short** data between a read-only property of an OLE control and a **short** data member.
DDX_OCText	Manages the transfer of **CString** data between a property of an OLE control and a **CString** data member.
DDX_OCTextRO	Manages the transfer of **CString** data between a read-only property of an OLE control and a **CString** data member.

If you're new to Microsoft Foundation Class (MFC) programming, or C++ programming in general, then there are probably many areas you find confusing or hard to understand. Even (or should I say "especially"?) veteran programmers have questions from time to time. If I had to pick one area where I receive the most inquiries, it would have to be Dialog Data Exchange (DDX) and Dialog Data Verification (DDV). Collectively, these mechanisms provide you a means to transfer information from your main application to and from a dialog box you wish to display. And although they may seem cryptic at first, spending the time to learn to use them will save you pain and agony in the long run!

First, how would we build a dialog in the days of 'C' and the Windows SDK? Well, we'd most likely insert a command into a menu, which would trigger a call to DialogBox(), or a derivative, in our command message handling code. One of the parameters passed to DialogBox() is a pointer to a subroutine we created to actually bring up the dialog box. If this dialog box required initialization, such as filling in default text in an edit box, we'd (probably) either hard-code the text into our WM_INITDIALOG handler or use a global variable as a text container. If the user modified the text, we'd have to write the new string back to the global variable or find some other viable means of retrieving the string prior to the dismissal of the dialog box.

Notes

Did u know? Because once the dialog box has been dismissed, all of its local variables are destroyed and no longer accessible

Ooh, there were some ugly words in that last paragraph. "Global variables"? Well, we did what we had to do to get our applications out the door on time. But today's Windows programmer can use the power of C++ and the MFC. Let's now think in terms of C++, then add some MFC concepts as they relate to a dialog box. In C++, when you instantiate an object from a class, the object exists until you specifically destroy it, either manually or by leaving your own current scope. So unlike a typical 'C' subroutine, our C++ class may have as many member functions (subroutines) and member variables ("global" to that class) as we like. Best of all, we can keep the object derived from the class around for as long as we need it. Now, suppose that for a minute we developed a class using MFC designed to display a dialog box. We could tell the object derived from the class to display itself, and when the user dismissed the dialog box, the C++ object would still exist, as we have not taken action to destroy it.

Notes MFC classes "contain" windows, or window handles, as if the class were a bottle…the class "holds" the window handle but is not the window itself.

So far, then, we know we can create a class to display a dialog box. Any class we create may contain member variables which we are free to initialize (assuming they are created as public). So, we build this dialog class, add some public member variables, and go to town. Right? Well, not quite yet. The problem we have is the dialog box, as a window, contains controls such as edit boxes, static text, radio buttons, etc. The class we instantiated merely holds a window handle, and even that is invalid until the dialog box itself is created. So how do we fill the controls inside the dialog box before it's created? We do so by using DDX.

Our first task on the road to using DDX is to create a main application, then create a new dialog box class. Assuming you've built your basic application (another topic, another day), you first "lay out" your dialog box using the dialog box editor (or resource editor, if you prefer) in Visual C++.

Example: Let's insert a new dialog box, IDD_MYDIALOG. Then, include a text edit box we'll call IDC_EDITBOX and a check box we'll call IDC_CHECKBOX (for the check box text, use "Check Me!"). Feel free to arrange these controls and give the dialog box any style or ornamentation you wish.

Now, activate the Class Wizard by selecting it under the "View" menu. We'll use the "Add Class" button under the "Message Map" tab—press this button and select "New". This will activate another dialog box, where we'll call our new class "DemoDialog" (under "Name") and give it a base class of CDialog (under "Base Class"). You should see the dialog identifier IDD_MYDIALOG "Dialog ID".

Caution To be sure Class Wizard saves your work to date, click OK twice. This will exit Class Wizard and store some new files for you…these are your dialog box class files.

Again activate Class Wizard, only this time select the "Member Variables" tab. Do you see your control identifiers, IDC_EDITBOXand IDC_CHECKBOX? You should. IDC_CHECKBOX will be

Notes

highlighted in the list box, so just press "Add Variable" and type in the variable name m_bIsChecked and press "OK".

Notes Class Wizard gave you a default variable type of boolean. To create a variable forIDC_EDITBOX, press "Add Variable" again and type the variable name m_strText. Class Wizard gave this variable a default class of CString.

In our main application, we need to modify our menu and/or toolbar to add a command to activate our dialog box. Assuming we've added a command to "Demo the Dialog" (again, another topic, another time), we'll add a message handler (in Class Wizard) to handle the user command to display the dialog box. Assuming the message handler is named "OnViewDemodialog", we would edit the handler to look like this:

```
void CMainFrame::OnViewDemodialog()
{
        // Let's set a CMainFrame boolean variable to use for our UI command
        // handling.
        m_bDemoDlgActive = TRUE;

        // Here, we create our dialog box class and initialize the
        // member variables.
        CString strResult;
        CDemoDialog MyDialog;

        MyDialog.m_strText = "Edit Me!"; // fill in edit box
        MyDialog.m_bIsChecked = FALSE; // set up check box

        if ( MyDialog.DoModal() == IDOK ) {
                // If the user pressed "OK", we'll stuff the results onto the
                // control bar.
                strResult.Format("(Returned: '%s','%s')",MyDialog.m_strText,
                        (MyDialog.m_bIsChecked ? "checked" : "unchecked"));
        } // if
        else {
                // User pressed "Cancel", so write to control bar.
                strResult = "(Canceled)";
        } // else

        // Update status bar
        m_wndStatusBar.SetWindowText(strResult); // always use for pane
0...
```

Notes

```
        // Reset our boolean.
        m_bDemoDlgActive = FALSE;
}
```

If you successfully compiled this and ran the resulting program, you could bring up the demonstration dialog box, modify the edit control, check the check box, and press "OK". What you would see would be the text you entered and the status of the check box displayed on the status bar (at the bottom of the window). The MFC DDX mechanism did all of the work for you! To you, working with your dialog box was as easy as working with another C++ class' member variables. Class Wizard is really something!

Task Illustrate the use of DDX_OCFloat.

Self Assessment

Fill in the blanks:

1. ..... function is used to manage the transfer of **BOOL** data between a read-only property of an OLE control and a **BOOL** data member.
2. ..... function is used to manage the transfer of **CString** data between a read-only property of an OLE control and a **CString** data member.
3. ..... function manages the transfer of **short** data between a property of an OLE control and a **short** data member.
4. One of the parameters passed to DialogBox() is a ..... to a subroutine we created to actually bring up the dialog box.
5. If the user modified the text, we'd have to write the new string back to the ..... variable or find some other viable means of retrieving the string prior to the dismissal of the dialog box.

12.2 Problems in Global Variables

As with all Heuristic Rules, this is not a rule that applies 100% of the time. Code is generally clearer and easier to maintain when it does not use globals, but there are exceptions. It is similar in spirit to GotoConsideredHarmful, although use of global variables is less likely to get you branded as an inveterate hacker.

12.2.1 Why Global Variables Should be Avoided when Unnecessary?

- ***Non-locality:*** Source code is easiest to understand when the scope of its individual elements are limited. Global variables can be read or modified by any part of the program, making it difficult to remember or reason about every possible use.
- ***No Access Control or Constraint Checking:*** A global variable can be get or set by any part of the program, and any rules regarding its use can be easily broken or forgotten. (In other words, get/set accessors are generally preferable over direct data access, and this is even more so for global data.) By extension, the lack of access control greatly hinders achieving security in situations where you may wish to run untrusted code (such as working with 3rd party plugins).

Notes

- ***Implicit coupling:*** A program with many global variables often has tight couplings between some of those variables, and couplings between variables and functions.

Did u know? Grouping coupled items into cohesive units usually leads to better programs.

- ***Concurrency issues:*** If globals can be accessed by multiple threads of execution, synchronization is necessary (and too-often neglected). When dynamically linking modules with globals, the composed system might not be thread-safe even if the two independent modules tested in dozens of different contexts were safe.
- ***Namespace pollution:*** Global names are available everywhere. You may unknowingly end up using a global when you think you are using a local (by misspelling or forgetting to declare the local) or vice versa. Also, if you ever have to link together modules that have the same global variable names, if you are lucky, you will get linking errors. If you are unlucky, the linker will simply treat all uses of the same name as the same object.
- ***Memory allocation issues:*** Some environments have memory allocation schemes that make allocation of globals tricky. This is *especially* true in languages where "constructors" have side-effects other than allocation (because, in that case, you can express unsafe situations where two globals mutually depend on one another). Also, when dynamically linking modules, it can be unclear whether different libraries have their own instances of globals or whether the globals are shared.
- ***Testing and Confinement:*** Source that utilizes globals is somewhat more difficult to test because one cannot readily set up a 'clean' environment between runs. More generally, source that utilizes global services of any sort (e.g. reading and writing files or databases) that aren't explicitly provided to that source is difficult to test for the same reason. For communicating systems, the ability to test system invariants may require running more than one 'copy' of a system simultaneously, which is greatly hindered by any use of shared services – including global memory – that are not provided for sharing as part of the test.

12.2.2 Why the Convenience of Global Variables sometimes Outweighs the Potential Problems?

- In a very small or one-off programs, especially of the 'plugin' sort where you're essentially writing a single object or short script for a larger system, using globals can be the simplest thing that works.
- When global variables represent facilities that truly are available throughout the program, their use simplifies the code.
- Some programming languages provide no support or minimal support for non-global variables.
- Some people jump through very complicated hoops to avoid using globals. Many uses of the Singleton Pattern are just thinly veiled globals. If something really should be a global, make it a global. Don't do something complicated because you *might* need it someday. If a global variable exists, I would assume that it is used. If it is used, there are methods associated with it. Even in the above cases, it's wise to consider using one of the *Alternatives to Global Variables* to control access to this facility. While this does help future-proof the code.

Notes

Example: When your 'small' program grows into a very large one, it can also simplify more immediate problems such as testing the code or making it work correctly in a concurrent environment.

Self Assessment

Fill in the blanks:

6. A global variable can be get or set by any part of the program, and any ..... regarding its use can be easily broken or forgotten.
7. If globals can be accessed by multiple threads of execution, ... is necessary (and too-often neglected).
8. When dynamically linking modules, it can be unclear whether different libraries have their own ..... of globals or whether the globals are shared.

12.3 Exchanging Data with a Dialog Box – Pointer Method

Dialog Box Data Exchange and Validation

The controls in a dialog box are specialized windows that store their own copy of the data that the user enters or changes. In the lifetime of a dialog box, you'll normally want to initialize these controls with data from your program, and then save that data back into those variables.

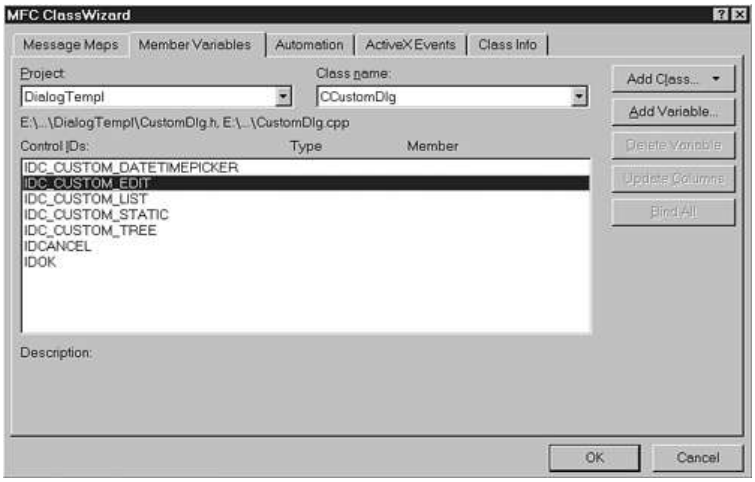
You'll probably also want to validate the values stored in the controls to ensure that they are within acceptable ranges when the user attempts to click OK to exit the dialog box.

Obviously, the first task in this process is to add new member variables to the dialog box handler class that correspond to the controls.

Mapping Member Variables to Controls

You can use ClassWizard to add member variables and provide the mapping for most of the dialog box controls via ClassWizard's Member Variables tab (see Figure 12.1).

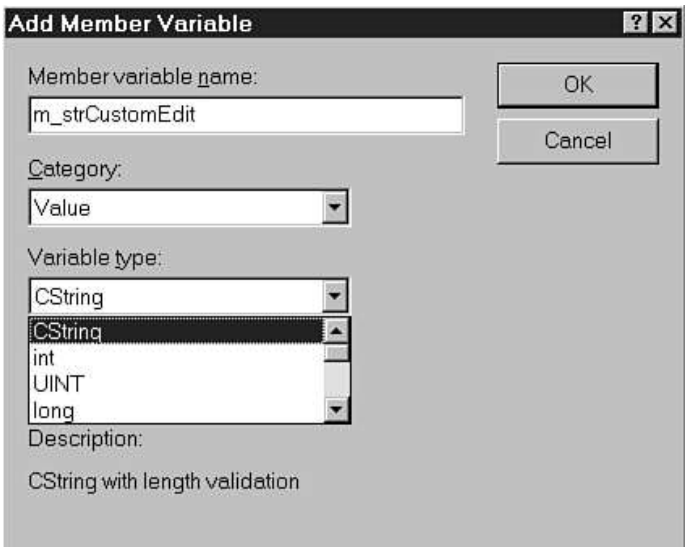
Figure 12.1: The ClassWizard Member Variables Tab



Notes

The Member Variables tab lists the control IDs of the various controls. You can select an ID and click the Add Variable button to display the Add Member Variable dialog box (see Figure 12.2).

Figure 12.2: The Add Member Variable Dialog Box



You'll notice that the Add Member Variable dialog box lets you set a category that can be a value or a control, as well as a variable type. The variable types available change depending on your selection in the Category combo box.

If you set the Category combo box to indicate value mapping, the Variable Type combo box lists member variable types that can be used with the type of control being mapped. These values are great for quick and easy value-oriented transfer, but often you'll need to map a control class to the control so that you can manipulate the control's more advanced features.

You can map a number of variables to the same control so that you can perform easy value transfer and allow control handler class mapping concurrently.

After you add the member variable map, you'll notice that the new member variable is inserted into your dialog box handler class definition between the ClassWizard AFX_DATA-generated comments.

The new variable also initializes your dialog box class's constructor function like this:

```
//{{AFX_DATA_INIT(CCustomDlg)
m_strCustomEdit = _T("");
//}}AFX_DATA_INIT
```

You'll see a new entry placed in the DoDataExchange() function like this:

```
void CCustomDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CCustomDlg)
      DDX_Text(pDX, IDC_CUSTOM_EDIT, m_strCustomEdit);
      //}}AFX_DATA_MAP
}
```

Notes

The new DDX_Text macro entry automates data transfer between the control identified by IDC_CUSTOM_EDIT and your m_strCustomEditmember variable.

If you were to add a control map for the edit control, you'd see a CEdit member variable entry added to the class definition, and a corresponding DDX_Control macro to map it to the control ID. Consider this example:

DDX_Control(pDX, IDC_CUSTOM_EDIT, m_editCustom);

If you insert other variable types, you'll see different DDX_ macros used to map the various types of controls to various data types. Table 12.1 lists some of these controls.

Table 12.1: Some Common Controls and Their Mapping Classes/Variables

Control	Mapping Class	Allowable Mapped Data Types
Static	CStatic	CString
Edit	CEdit	CString, DWORD, UINT, int, long
		double, float, BOOL, short
		COleDateTime & ColeCurrency
Button	CButton	None
CheckBox	CButton	BOOL
3-State CheckBox	CButton	int
Radio	CButton	int
ListBox	CListBox	CString, int
ComboBox	CComboBox	CString, int
Extended Combo	CComboBoxEx	CString, int
ScrollBar	CScrollBar	int
Spin	CSpinButtonCtrl	None
Progress Bar	CProgressCtrl	None
Slider Control	CSliderCtrl	int
List Control	CListCtrl	None
Tree Control	CTreeCtrl	None
Date Time Picker	CDateTimeCtrl	CTime, COleDateTime
Month Calendar	CMonthCalCtrl	CTime, COleDateTime

You can add some simple validation maps to certain controls, such as Edit controls. Depending on the variable type mapped, the lower section of the Member Variables tab displays a section that lets you specify validation information.

If you map a CString to an Edit control, for example, you can set the validation rules to limit the maximum number of characters allowed in the Edit control. If you map an integer to the Edit control, you can set upper and lower ranges for the entered value.

If you set any of these validation rules, ClassWizard adds DDV_ routines to the DoDataExchange() function to perform validation, like this:

DDV_MaxChars(pDX, m_strCustomEdit, 10);

Several different DDV_ routines exist for the various types of validation rules, member variables, and control types.

Notes

The Data Exchange and Validation Mechanism

The DoDataExchange() function is called several times during the lifetime of a dialog box and performs a variety of tasks. When the dialog box is initialized, this function subclasses any mapped controls through the DDX_Control routine (discussed in more detail later in the section, "Initializing the Dialog Box Controls"). Then it transfers the data held in the member variables to the controls using the DDX_ routines. Finally, after the user clicks OK, the data from the controls is validated using DDV_ routines and then transferred back into the member variables using the DDX_ routines again.

You'll notice that the DoDataExchange() function is passed a pointer to a CDataExchange object. This object holds the details that let the DDX routines know whether they should be transferring data to or from the controls. The DDX_ routines then implement the Windows message required to set or retrieve data from the control associated with the given control ID.

When the m_bSaveAndValidate member is set to TRUE, the data exchange should transfer data from the controls to the member variables and perform validation. It is set to FALSE when data from the member variables should be loaded into the controls. You can add your own custom code to DoDataExchange() to transfer data to or from the controls and check the m_bSaveAndValidatemember of the CDataExchange object to see whether you should be transferring the data to or from the control.

12.4 Modal, Modeless and System Modal Dialog boxes

Dialog boxes are specifically used for accepting input from the user or displaying a message to the user. Windows operating system comes with default dialog boxes like Font dialog box, Printer dialog box, Color dialog box etc. Most of the applications specially those vendored by Microsoft borrow these dialog boxes from Windows whenever they need. There are three kinds of dialog boxes.

1. Modal Dialog Box (Application Modal Dialog Box)
2. Modeless Dialog Box
3. System Modal Dialog Box

12.4.1 Modal Dialog Box

Modal dialog boxes can be also called as Application Modal dialog box. These dialog boxes insist you to respond to them before continuing in the same application. A Modal dialog box while in action, stops running your code until it is closed or hidden. Although, you can call default Dialog box as a Modal dialog box, but you can also create of your own. The Show method of form uses VbModal style to load the form as a Modal dialog box. The general syntax you follow will be :FormName.Show VbModal

12.4.2 Modeless Dialog Box

Modeless dialog boxes do not need to be closed to loose their focus. They are just like any other form in your application and loose their focus as soon as you click some other Window outside the application.

Example: A Find and Replace dialog box, Document Windows in Word application are few examples of Modeless dialog boxes.

Notes

A Form is by default loaded in this state if no arguments are supplied with Show method.

12.4.3 System Modal Dialog Box

A System Modal dialog box restricts the user from continuing work on the system unless it gets unloaded.

Example: Screen Saver with passwords is the pretty example of a Modal dialog box. Creating such a dialog box is a bit tricky, however you can have a default message box in System Modal mode.

The show method of forms does not give you such style. In order to call a default message box in a system Modal style, use the following syntax:

MsgBox "Message String", VbOKonly + VbSystemModal, "Title"

Self Assessment

Fill in the blanks:

9. Dialog boxes are specifically used for accepting ..... from the user or displaying a message to the user.
10. Modal dialog boxes can be also called as ..... .
11. The ..... method of form uses VbModal style to load the form as a Modal dialog box.
12. ..... dialog boxes are just like any other form in your application and loose their focus as soon as you click some other Window outside the application.
13. A ..... dialog box restricts the user from continuing work on the system unless it gets unloaded.

12.5 Creating a Modeless Dialog Box

You create a modeless dialog box by means of the CreateDialog function, mentioning the identifier or name of a dialog box template resource and a pointer to the dialog box procedure. CreateDialog loads the template, creates the dialog box, and optionally exhibits it.

Caution Your application is accountable for taking and transmitting user input messages to the dialog box procedure.

Example: In the following example, the application exhibits a modeless dialog box — if it is not already shown — when the user clicks Go To from an application menu. The dialog box includes an edit control, a check box, and OK and Cancel buttons. The dialog box template is a resource in the application's executable file and has the resource identifier DLG_GOTO. The user enters a line number in the edit control and verifies the check box to state that the line number is relative to the current line. The control identifiers are ID_LINE, ID_ABSREL, IDOK, and IDCANCEL.

The statements in the first part of the example create the modeless dialog box. These statements, in the window procedure for the application's main window, create the dialog box when the

Notes

window procedure obtains a **WM_COMMAND** message having the IDM_GOTO menu identifier, but only if the global variable does not already enclose a valid handle. The second part of the example is the application's main message loop. The loop involves the **IsDialogMessage** function to make sure that the user can utilize the dialog box keyboard interface in this modeless dialog box. The third part of the the dialog box procedure. The procedure obtains the contents of the edit control and check box when the user clicks the **OK** button. The procedure demolishes the dialog box when the user clicks the **Cancel** button.

```
HWND hwndGoto = NULL;   // Window handle of dialog box


...


case WM_COMMAND:
      switch (LOWORD(wParam))
     {
          case IDM_GOTO:
               if (!IsWindow(hwndGoto))
             {
                   hwndGoto = CreateDialog(hinst,
                                              MAKEINTRESOURCE(DLG_GOTO),
                                            hwnd,
                                              (DLGPROC)GoToProc);
                   ShowWindow(hwndGoto, SW_SHOW);
             }
             break;
     }
     return 0L;
```

In the former statements, **CreateDialog** is called only if hwndGoto does not enclose a valid window handle. This makes sure that the application does not exhibit two dialog boxes at the same time. To support this method of checking, the dialog procedure must set to **NULL** when it demolishes the dialog box.

The message loop for an application includes the following statements:

```
BOOL bRet;


while ((bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
{
     if (bRet == -1)
     {
          // Handle the error and possibly exit
     }
      else if (!IsWindow(hwndGoto) || !IsDialogMessage(hwndGoto, &msg))
     {
```

Notes

```
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
```

The loop verifies the validity of the window handle to the dialog box and only calls the **IsDialogMessage** function if the handle is valid. IsDialogMessage only processes the message if it relates to the dialog box. Or else, it returns **FALSE** and the loop transmits the message to the suitable window.

The following statements identify the dialog box procedure.

```
int iLine;                // Receives line number.
BOOL fRelative;           // Receives check box status.


BOOL CALLBACK GoToProc(HWND hwndDlg, UINT message, WPARAM wParam, LPARAM
lParam)
{
     BOOL fError;


     switch (message)
    {
          case WM_INITDIALOG:
                CheckDlgButton(hwndDlg, ID_ABSREL, fRelative);
            return TRUE;


          case WM_COMMAND:
               switch (LOWORD(wParam))
             {
                  case IDOK:
                         fRelative = IsDlgButtonChecked(hwndDlg, ID_ABSREL);
                        iLine = GetDlgItemInt(hwndDlg, ID_LINE, &fError,
                      fRelative);
                      if (fError)
                     {
                             MessageBox(hwndDlg, SZINVALIDNUMBER, SZGOTOERR,
                         MB_OK);
                            SendDlgItemMessage(hwndDlg, ID_LINE, EM_SETSEL, 0,
                       -1L);
                     }
                     else


                       // Notify the owner window to carry out the task.


                      return TRUE;
```

Notes

```
                case IDCANCEL:
                     DestroyWindow(hwndDlg);
                    hwndGoto = NULL;
                    return TRUE;
            }
    }
     return FALSE;
}
```

In the former statements, the procedure processes the WM_INITDIALOG and WM_COMMAND messages.

Task What is the job of **IsDialogMessage** function? Discuss.

Self Assessment

Fill in the blanks:

14. ..... loads the template, creates the dialog box, and optionally exhibits it.
15. The procedure demolishes the dialog box when the user clicks the ..... button.

12.6 Summary

- There are various DDX_OC functions used to exchange data between a property of an OLE control in a dialog box, form view, or control view object and a data member of the dialog box, form view, or control view object.
- When we insert a command into a menu, it would trigger a call to DialogBox(), or a derivative, in our command message handling code.
- In our main application, we need to modify our menu and/or toolbar to add a command to activate our dialog box.
- Global variables can be read or modified by any part of the program, making it difficult to remember or reason about every possible use.
- Modal dialog boxes (also called as Application Modal dialog box) insist you to respond to them before continuing in the same application.
- Modeless dialog boxes are just like any other form in your application and loose their focus as soon as you click some other Window outside the application.
- A System Modal dialog box restricts the user from continuing work on the system unless it gets unloaded.
- You create a modeless dialog box by means of the **CreateDialog** function, mentioning the identifier or name of a dialog box template resource and a pointer to the dialog box procedure.

Notes

12.7 Keywords

CreateDialog: CreateDialog loads the template, creates the dialog box, and optionally exhibits it.

Modal Dialog Boxes: Modal dialog boxes (also called as Application Modal dialog box) insist you to respond to them before continuing in the same application.

Modeless Dialog Boxes: Modeless dialog boxes are just like any other form in your application and loose their focus as soon as you click some other Window outside the application.

System Modal Dialog Box: A System Modal dialog box restricts the user from continuing work on the system unless it gets unloaded.

12.8 Review Questions

1. Explain the process of exchanging data with a dialogue box using Global Variable Method and Pointer Method.
2. Illustrate various DDX_OC functions used to exchange data.
3. How DDX is used to create a new dialog box class? Explicate.
4. What are the problems that appear while using global variables? Discuss.
5. Why the convenience of global variables sometimes outweighs the potential problems?
6. Illustrate the function of Application Modal Dialog Box.
7. What is system modal dialog box? Also write the syntax used in system modal dialog box.
8. Make distinction between modal and modal dialog boxes. Explain with appropriate examples.
9. Elucidate how to create a Modeless Dialog Box a Modeless Dialog Box.
10. What are the memory allocation issues related with global variables? Illustrate.

Answers: Self Assessment

1. DDX_OCBoolRO
2. DDX_OCTextRO
3. DDX_OCShort
4. pointer
5. global
6. rules
7. synchronization
8. instances
9. input
10. Application Modal dialog box
11. Show
12. Modeless
13. System Modal
14. CreateDialog
15. Cancel

12.9 Further Readings

Notes

Books

Brent E. Rector, *Win32 Programming*, Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

www.functionx.com/win32/Lesson04.htm

Notes

Unit 13: Windows GDI

CONTENTS

Objectives

Introduction

13.1 Windows GDI

13.1.1 Displaying Text

13.1.2 Displaying Pixels

13.1.3 Drawing Lines

13.1.4 Drawing Filled Shapes

13.1.5 Pens and Brushes

13.1.6 Window Size

13.1.7 Forcing a Redraw

13.2 Summary

13.3 Keywords

13.4 Review Questions

13.5 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain Windows GDI
- Discuss the Windows GDI capabilities

Introduction

The Microsoft Windows graphics device interface (GDI) enables applications to use graphics and formatted text on both the video display and the printer. Windows-based applications do not access the graphics hardware directly. Instead, GDI interacts with device drivers on behalf of applications.

- ***Where applicable:*** GDI can be used in all Windows-based applications.
- ***Developer audience:*** This API is designed for use by C/C++ programmers. Familiarity with the Windows graphical user interface and message-driven architecture is required.
- ***Run-time requirements:*** For information, on which operating systems are required to use a particular function.

13.1 Windows GDI

GDI stands for Graphics Device Interface. It provides many functions for displaying graphics in your Windows application.

Notes

Notes In GDI calls screen position 0,0 is the top left of the window. If you need to work out the window width and height follow this link here: Window size

An example application showing these GDI functions can be downloaded here: WinGDIdemo.exe. In addition the source code for this demo can also be downloaded: WinGDIDemoCode.zip

The different GDI capabilities:

- Displaying Text
- Displaying Pixels
- Drawing Lines
- Drawing Filled Shapes
- Pens & Brushes
- Window size
- Forcing a redraw

13.1.1 Displaying Text

To display some text you can use the TextOut function. In order to use this you need to obtain a handle to the device, an HDC and release it after use.

To obtain the handle you call: BeginPaint, it takes your window handle and a pointer to a PAINTSTRUCT that you have declared. This structure is just used by windows so you don't need to do anything apart from pass it on to functions.

The definition of the TextOut function is:

BOOL TextOut(HDC hdc, int x, int y, LPCSTR lpString, int cbString);

It takes the handle you were returned from BeginPaint, the x, y position on screen, a pointer to some text and the number of characters you want to display.

After displaying your text you must call EndPaint to release the handle.

For example, to display 'Hello World' at screen position 10,10 you could do this:

```
PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

TextOut(hdc,10,10,"Hello World",11);

EndPaint(hWnd, &ps);
```

Did u know? **What is GDI +?**

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.

Notes

13.1.2 Displaying Pixels

You can drawn single pixels to the screen using the following function:

SetPixel(HDC hdc, int X, int Y, COLORREF crColor);

Again it requires a handle to a device context (HDC) and a screen position. Additionally the colour you want to set the pixel to can be provided. This needs to be given in a Win32 type COLORREF. Fortunately, we can use a macro (RGB) to convert from three Red, Green and Blue values to a COLORREF type. Each colour component can range from 0 to 255.

To display a red pixel at screen co-ordinates 100,100 we would write:

SetPixel(hdc, 100, 100, RGB(255,0,0));

Task In a group of four analyze why GDI can be used in all Windows-based applications.

13.1.3 Drawing Lines

There are a number of functions for drawing lines using GDI. You can use MoveTo and LineTo to position a pen on the screen and draw a line to another position. You can also use PolyLine, PolyPolyLine, Arc, etc. To explore which ones are available look in the MSDN help. We will describe here the use of PolyLine.

PolyLine draws lines between points defined in an array you pass to the function. The function definition is:

Polyline(HDC hdc, CONST POINT *lppt, int cPoints);

This takes the device handle, an array of points and how many points there are in the array. It draws lines between each point. POINT is a structure with member variables x and y.

To draw a simple diagonal line we could do this:

```
POINT pntArray[2];

pntArray[0].x=10;

pntArray[0].y=10;

pntArray[1].x=100;

pntArray[1].y=100;

Polyline(hdc, pntArray, 2);
```

Caution The color and style of the line can be changed by using pens.

13.1.4 Drawing Filled Shapes

You can draw filled rectangles using FillRect and filled ellipses using Ellipse. Also, not covered here, you can draw filled polygons, pies, rounded rectangles etc. Again look in the MSDN help for details.

Notes

To draw a filled ellipse we can use:

BOOL Ellipse(HDC hdc, int nLeftRect, int nTopRect, int nRightRect, int nBottomRect);

This function takes the device context and the screen positions of a rectangle that represents the bounding rectangle for the ellipse. The ellipse will be drawn to fit in this rectangle.

To draw a filled rectangle we can use:

Example: FillRect(HDC hDC, CONST RECT *lprc, HBRUSH hbr);

This function takes the device context, a rectangle structure and a brush. The RECT structure contains member variables left, right, top and bottom. Brushes are described in the next section: Pens and Brushes.

13.1.5 Pens and Brushes

GDI uses the concept of Pens and Brushes. A Pen defines the style and colour that pixels will be drawn in while a brush determines the fill colour of shapes. A number of the GDI drawing functions can be effected by the pen or brush chosen.

Pens

To create a pen we need to obtain a handle to one. This handle is named HPEN, we can keep hold of this handle during the lifetime of our program but must remember to delete it before exiting.

To create a pen we use the function:

HPEN CreatePen(int fnPenStyle, int nWidth, COLORREF crColor);

This function takes a style, a pen width and a colour. The style must be one of the predefined styles. The main ones are shown below:

PS_SOLID - Pen is solid.

PS_DASH - Pen is dashed.

PS_DOT - Pen is dotted.

PS_DASHDOT - Pen has alternating dashes and dots.

PS_DASHDOTDOT - Pen has alternating dashes and double dots.

PS_NULL - Pen is invisible.

So to create a solid green pen of 1 pixel width we would write:

HPEN greenPen=CreatePen(PS_SOLID, 1, RGB(0,255,0));

It is a good idea to create our pens (and brushes) in advance of drawing and then apply them as required. To apply a pen so future drawing commands use it we must select it. This is known as selecting it into the device context and is achieved by using SelectObject. One thing we must be aware of is that when we no longer want to use our pen we need to re-select the old pen. Luckily SelectObject returns the old pen. So to use our green pen to draw a line we may do this:

// Select our green pen into the device context and remember previous pen

HGDIOBJ oldPen=SelectObject(hdc,greenPen);

// Draw our line

Notes

```
Polyline(hdc, pntArray, 2);
// Select the old pen back into the device context
SelectObject(hdc,oldPen);
```

Brushes

Creating and using Brushes is very similar to pens. We can use CreateSolidBrush to create a solid coloured brush or CreateBrushIndirect to create a brush with specific styles (like hatched) or even using a loaded bitmap. You can use a bitmap as a repeating pattern for your brush using CreatePatternBrush. Here we will describe CreateSolidBrush for the others please look int he MSDN help file.

```
HBRUSH CreateSolidBrush( COLORREF crColor)
```

This is a very simple function that takes the required colour and returns a handle to a brush. We can use it in much the same way as the pen. To create a blue brush we would write:

```
HBRUSH blueBrush=CreateSolidBrush(RGB(0,255,0));
```

To use it to fill a rectangle:

```
RECT rct;
rct.left=10;
rct.right=100;
rct.top=10;
rct.bottom=200;
FillRect(hdc, &rct, blueBrush);
```

13.1.6 Window Size

When you create a window you can specify its size. However the size of the window does not always relate to the drawing area as there are menu bars and borders that also take up room. Additionally the user can simply alter the size of your window at any time. So we need a way of determining the current drawing area. We do this using the GetClientRect call. This returns a rectangle defining the area of the window your program (the client) can draw into. You must pass in the handle to your window and the address of a rectangle that will be filled by the function.

Example: RECT clientRect;

```
GetClientRect(hWnd,&clientRect);
```

13.1.7 Forcing a Redraw

As we have mentioned previously you can only draw your window when Windows tells you to via a WM_PAINT message. Sometimes you would like to update your window whenever you want. To do this you have to tell Windows that your window is dirty and needs a redraw. You can do this by using InvalidateRect. This tells Windows your window is dirty and needs redrawing. Windows then sends you a WM_PAINT message telling you to draw. InvalidateRect takes three parameters, the handle to your window, the rectangular area of your window that

Notes

needs to be redrawn (or NULL for the whole window) and a Boolean indicating if you wish the window to be cleared first. So the standard call for clearing the whole window is:

InvalidateRect(hWnd,NULL,TRUE);

Self Assessment

Fill in the blanks:

1. To display some text you can use the …… function.
2. To obtain the handle you call: BeginPaint, it takes your window handle and a pointer to a …….. that you have declared.
3. We can use a macro (RGB) to convert from three Red, Green and Blue values to a …….. type.
4. Additionally the colour you want to set the …….. to can be provided.
5. You can use ……. and LineTo to position a pen on the screen and draw a line to another position.
6. …… draws lines between points defined in an array you pass to the function.
7. Additionally the user can simply alter the ……. of your window at any time.
8. You can use a bitmap as a repeating pattern for your brush using …….
9. A …….. defines the style and colour that pixels will be drawn in while a brush determines the fill colour of shapes.
10. The …….. structure contains member variables left, right, top and bottom.

Caselet

Banking on BlackBerry

There's this old joke about two campers who spy a grizzly coming towards them from afar. One of them immediately takes off his shoes and substitutes them for runners. "Why are you doing that? You can't outrun that bear even with running shoes," said his friend, to which he replied, "Who cares about the grizzly? All I have to do is outrun you."

Employ this logic in the real world, and you see a similar situation with mobile manufacturers who are busy trying to woo users, and have to keep developers on their side. Because if you can't attract developers, you won't get any great apps, and without apps, your smartphone isn't worth the pixels on its screen.

Rise of the Underdog?

In this game, BlackBerry is at a slight disadvantage because it is not perceived to be a big player, and so has to strive a little harder to keep up.

Pointing this out, Anil Pai, Mobile Security Analyst, TCS, says, "Android and iPhone are the market leaders, followed by Windows. BlackBerry is in the fourth spot and may pick up in future." Pai says that the launch of the QNX operating system for BlackBerry in a few

Contd...

Notes

months, coupled with the security features the manufacturer offers, will make it a viable alternative to the other three. And while TCS is currently not working on anything for BlackBerry platform from the security standpoint, he said that the company plans to do this in the near future. Getting a big software player like TCS to work on its platform is a big plus for BlackBerry.

This is because the stakes are huge. According to an IDC report released in January 2012, the world's mobile worker population will reach 1.3 billion by 2015, representing 37.2 per cent of the total workforce. Asia Pacific (excluding Japan) will see the maximum growth from around 601.7 million mobile workers in 2010 to 838.7 million in 2015, spurred primarily by India and China.

Many of them will be carrying smartphones and being the number fourth in this list, one expects BlackBerry to try that much harder to woo developers because apps hold the key to smartphone adoption.

Ironing Out the Creases

One of the things that might help BlackBerry could be its native functionality, said Sunil Mishra, Senior Software Engineer, Creative Commons, who has been developing apps on both Android and BlackBerry.

"Both are good, but BlackBerry has a richer user interface." Mishra said that he would continue to develop apps for BlackBerry in spite of his knowledge of Android because, as he put it, "Android is easy to learn but difficult to debug."

Such news will be music to RIM's ears, which will no doubt want to increase its market share in India – a tough thing to do if you are not a major player because the market for smartphones is rather limited in India as of now. In November last year, Gartner said that while Indian mobile handset sales would reach 231 million units in 2012, an increase of 8.5 per cent over 2011 sales of 213 million units, very few would be smartphones. In fact, in the first three quarters of the calendar year 2011, smartphone sales in India made up 6 per cent of total device sales, and this is expected to increase only to 8 per cent in 2012. View this statistic in a different light, and this translates to about 18.48 million smartphones being shipped in India in calendar year 2012, with BlackBerry taking a relatively smaller percentage of sales.

Another big problem with BlackBerry is the total number of apps available. According to Annie Mathew, Head of Alliances and Developer Relations, India, Research in Motion, there are about 50,000 apps on BlackBerry's App World, which is a very small number when compared to Apple, which has at least 5 lakh apps on its App Store. But BlackBerry is trying to turn around its smaller base by offering a customised service to developers, she says. Citing an example, she says that when Dhingana, which was launched on the App Store two years ago, first came to BlackBerry, they had a lot of issues, but BlackBerry's team helped them to resolve them. "It is important for developers that they should find it easy to develop apps," she said.

And some developers are finding this to be true. Vineeth Karunakaran, Senior Software Engineer, USD Global, who has been developing apps for BlackBerry for three years (he also develops J2ME apps for Nokia and other platforms, besides writing apps for Symbian) says that he shifted to developing for BlackBerry because of the satisfaction he gets from the platform. "We are able to do everything we want. It is better than offerings from competitors," he said.

Contd...

Notes

Language Barriers

Of course, there is also another issue – programming languages. Once a developer learns a programming language, he is not always eager to jump from what he knows to what he doesn't because this involves relearning a language. Karunakaran, who is well-versed with Java, says that he can't move to iOS because he is required to program it in Objective C, a language that he is not familiar with. He also has no desire to move to Android, a platform that he says he doesn't like. "Android has a lot of bugs even with basic features like phone calls and SMS. I have an Android phone and these bugs are making it difficult for me to use the phone," he said.

So, is the BlackBerry developer market filled with people who don't like other platforms, or can't move away for various reasons? Mathew disagrees. "Apple is not the most popular brand in India. That is why developers want to get on to BlackBerry." Highlighting Indian players like MakeMyTrip, Naukri and Shaadi, she said, "They first made an appearance on BlackBerry and only then did they get on other platforms. For developers in India, it is BlackBerry, Android and iOS, in that order."

13.2 Summary

- GDI stands for Graphics Device Interface. It provides many functions for displaying graphics in your Windows application.
- GDI uses the concept of Pens and Brushes. A Pen defines the style and colour that pixels will be drawn in while a brush determines the fill colour of shapes.
- We can use CreateSolidBrush to create a solid coloured brush or CreateBrushIndirect to create a brush with specific styles (like hatched) or even using a loaded bitmap.
- Sometimes you would like to update your window whenever you want. To do this you have to tell Windows that your window is dirty and needs a redraw. You can do this by using InvalidateRect.

13.3 Keywords

GDI: Graphics Device Interface

HDC: Handle to a Device Context

13.4 Review Questions

1. Windows-based applications do not access the graphics hardware directly. Explain.
2. To obtain the handle you call: BeginPaint, it takes your window handle and a pointer to a PAINTSTRUCT that you have declared. Discuss.
3. You can also use PolyLine, PolyPolyLine, Arc etc. To explore which ones are available look in the MSDN help. Analyze.
4. A number of the GDI drawing functions can be effected by the pen or brush chosen. Elaborate.
5. When you create a window you can specify its size. Explain with an example.

Notes

Answers: Self Assessment

1.	TextOut	2.	PAINTSTRUCT
3.	COLORREF	4.	pixel
5.	MoveTo	6.	PolyLine
7.	size	8.	CreatePatternBrush
9.	Pen	10.	RECT

13.5 Further Readings

Books

Beginning C: From Novice to Professional, Fourth Edition by Ivor Horton.

C# Programming Language, The (2nd Edition) (Microsoft .NET Development Series) (Hardcover) by Anders Hejlsberg, Scott Wiltamuth, Peter Golde, Publisher: Addison-Wesley Professional; 2 edition (June 9, 2006)

CLR via C#, Second Edition (Paperback) by Jeffrey Richter, Publisher: Microsoft Press; 2nd edition (February 22, 2006)

Essential C# 2.0 (Microsoft .NET Development Series) (Paperback) by Mark Michaelis, Publisher: Addison-Wesley Professional (July 13, 2006)

Expert C# 2005 Business Objects, Second Edition by Rockford Lhotka.

Pro C# 2005 and the .NET 2.0 Platform, Third Edition by Andrew Troelsen.

Programming in the Key of C# (Paperback) by Charles Petzold, Publisher: Microsoft Press; 1 edition (July 30, 2003)

Online links

http://msdn.microsoft.com/en-us/library/dd145203(v=vs.85).aspx

http://en.wikipedia.org/wiki/Graphics_Device_Interface

Unit 14: Text and Graphics Output

Notes

CONTENTS

Objectives
Introduction
14.1 Character Mode vs Graphic Mode
14.1.1 Character Mode
14.1.2 Graphic Mode
14.2 The Device Context
14.3 Text Output
14.4 WM_PAINT Message
14.5 Changing the Device Context
14.6 Device Context Settings
14.6.1 Setting and Retrieving the Device Context Brush Color Value
14.6.2 Setting the Pen or Brush Color
14.7 Graphics Output
14.7.1 Line and Shape Controls
14.7.2 The Image Box and the Picture Box
14.8 Animated Graphics
14.9 The Peek Message [] Loop
14.9.1 The GetMessage()
14.9.2 A New Function, PeekMessage()
14.10 Summary
14.11 Keywords
14.12 Review Questions
14.13 Further Readings

Objectives

After studying this unit, you will be able to:

- Understand Character mode vs. Graphic Mode
- Discuss the concept of Device Context
- Discuss Text output and WM_PAINT message
- Understand Graphics Output
- Illustrate the concept of changing the device context

Notes

- Discuss Animated Graphics
- Understand the function of Peek Message [] Loop

Introduction

In this unit, you will understand various concepts related to text and graphics output such as Character mode vs. Graphic Mode, Device Context, Text output, WM_PAINT message, Graphics Output, Animated Graphics, etc.

14.1 Character Mode vs. Graphic Mode

14.1.1 Character Mode

Many video adapters support several different modes of resolution. All such modes are divided into two general categories: *character mode* (also called *text mode*) and *graphics mode*. In character mode, the display screen is treated as an array of blocks, each of which can hold one ASCII character. Of the two modes, character mode is much simpler. Programs that run in character mode generally run much faster than those that run in graphics mode, but they are limited in the variety of fonts and shapes they can display. Programs that run entirely in character mode are called *character-based programs*.

14.1.2 Graphic Mode

In graphics mode, the display screen is treated as an array of pixels, with characters and other shapes formed by turning on combinations of pixels. Of the two modes, graphics mode is the more sophisticated. Programs that run in graphics mode can display an unlimited variety of shapes and fonts, whereas programs running in character mode are severely limited. Programs that run entirely in graphics mode are called *graphics-based* programs.

Self Assessment

Fill in the blanks:

1. Programs that run entirely in character mode are called ..... programs.
2. In graphics mode, the display screen is treated as an array of ..... .

14.2 The Device Context

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.

CPaintDC objects encapsulate the common idiom of Windows, calling the BeginPaint function, then drawing in the device context, then calling the EndPaint function. The CPaintDC constructor calls BeginPaint for you, and the destructor calls EndPaint. The simplified process is to create the CDC object, draw, and destroy the CDC object. In the framework, much of even this process is automated. In particular, your OnDraw function is passed a CPaintDC already prepared (via OnPrepareDC), and you simply draw into it. It is destroyed by the framework and the

Notes

underlying device context is released to Windows upon return from the call to your OnDraw function.

CClientDC objects encapsulate working with a device context that represents only the client area of a window. The CClientDC constructor calls the GetDC function, and the destructor calls the ReleaseDC function. CWindowDC objects encapsulate a device context that represents the whole window, including its frame.

Did u know? CMetaFileDC objects encapsulate drawing into a Windows metafile.

Caution In contrast to the CPaintDC passed to OnDraw, you must in this case call OnPrepareDC yourself.

Task Illustrate the function of CClientDC objects.

Self Assessment

Fill in the blanks:

3. A ..... is a Windows data structure containing information about the drawing attributes of a device such as a display or a printer.
4. The ..... constructor calls BeginPaint for you, and the destructor calls EndPaint.
5. CWindowDC objects encapsulate a device context that represents the whole window, including its frame.

14.3 Text Output

Text	Output
create_interactive_index	In a Textflow define some terms to be indexed and create a sorted index from the indexed terms.
vertical_alignment_in_fitbox	Control the vertical alignment of text in the fitbox.
keep_lines_together	Control the lines kept together on the page.
drop_caps	Create an initial drop cap at the beginning of some text.
distance_between_paragraphs	Control the distance between adjacent paragraphs.
avoid_linebreaking	Create a Textflow and define various options for line breaking.
bulleted_list	Output numbered and bulleted lists.
continue_note_after_text	Insert a dot sequence automatically at the end of a textflow fitbox after the last word which can be showed together with the dots completely inside the fitbox.
current_text_position	Demonstrate how the current text position can be used to output simple text, text lines, or Textflows next to one another.
dot_leaders_with_tabs	Use leaders to fill the space defined by tabs between left-aligned and right-aligned text, such as in a table of contents.
footnotes_in_text	Create footnotes (superscript text) in a Textflow provided with links to jump to the footnote text.

Contd...

Notes

image_as_text_fill_color	Create outline text and fill the interior of the glyphs with an image.
invisible_text	Output invisible text on top of an image.
leaders_in_textline	Use dot leaders to fill the space between text and a page number such as in a table of contents.
numbered_list	Output numbered lists with the numbers left- or right-aligned.
process_utf8	Read text in the UTF-8 format and output it.
rotated_text	Create text output which does not run horizontally, but at some angle.
shadowed_text	Create a shadowed text line with the shadow option of fit_textline.
simple_stamp	Create a stamp across the page which runs diagonally from one edge to the other.
starter_textflow	Create multi-column text output which may span multiple pages.
starter_textline	Demonstrate various options for placing a text line.
tabstops_in_text	Create a simple multi-column layout using tab stops.
text_as_clipping_path	Output text filled with an image.
text on a path	Create text on a path.
text_on_color	Place a text line and a Textflow on a colored background.
text_with_image_clipping_path	Use the clipping path from a TIFF or JPEG image to shape text output.
transparent_part_of_text	Use gstate in Textflow, e.g. for transparency/opacity
transparent_text	Create some transparent text.
underlined_text	Create underlined text.
weblink_in_text	Create a Textflow and integrate colorized Web links in the text.
wrap_text_around_images	Place images within a Textflow.
wrap_text_around_polygons	Use arbitrary polygons as wrapping shapes for text to wrap around.

These text outputs are used in Windows GDI which is explained in Unit 13.

Self Assessment

Fill in the blank:

6. ..... is used to create an initial drop cap at the beginning of some text.

14.4 WM_PAINT Message

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.

A window receives this message through its **WindowProc** function.

```
LRESULT  CALLBACK  WindowProc(
   HWND  hwnd,
   UINT   uMsg,
```

Notes

```
   WPARAM  wParam,
   LPARAM  lParam
);
```

Parameters

wParam: This parameter is not used.

lParam: This parameter is not used.

Return Value

An application returns zero if it processes this message.

The **WM_PAINT** message is generated by the system and should not be sent by an application. To force a window to draw into a specific device context, use the **WM_PRINT** or **WM_PRINTCLIENT** message. Most common controls support the **WM_PRINTCLIENT** message.

Caution This requires the target window to support the **WM_PRINTCLIENT** message.

The **DefWindowProc** function validates the update region. The function may also send the **WM_NCPAINT** message to the window procedure if the window frame must be painted and send the **WM_ERASEBKGND** message if the window background must be erased.

The system sends this message when there are no other messages in the application's message queue. **DispatchMessage** determines where to send the message; **GetMessage** determines which message to dispatch. **GetMessage** returns the **WM_PAINT** message when there are no other messages in the application's message queue, and **DispatchMessage** sends the message to the appropriate window procedure.

Notes A window may receive internal paint messages as a result of calling **RedrawWindow** with the RDW_INTERNALPAINT flag set. In this case, the window may not have an update region. An application should call the **GetUpdateRect**function to determine whether the window has an update region. If **GetUpdateRect** returns zero, the application should not call the **BeginPaint** and EndPaint functions.

Task What does **dispatchMessage** signify?

Self Assessment

Fill in the blanks:

7. The ..... message is sent when the system or another application makes a request to paint a portion of an application's window.
8. ..... determines where to send the message.

Notes

14.5 Changing the Device Context

The following functions are used with device contexts.

Function	Description
CancelDC	Cancels any pending operation on the specified device context.
ChangeDisplaySettings	Changes the settings of the default display device to the specified graphics mode.
ChangeDisplaySettingsEx	Changes the settings of the specified display device to the specified graphics mode.
CreateCompatibleDC	Creates a memory device context compatible with the specified device.
CreateDC	Creates a device context for a device using the specified name.
CreateIC	Creates an information context for the specified device.
DeleteDC	Deletes the specified device context.
DeleteObject	Deletes a logical pen, brush, font, bitmap, region, or palette, freeing all system resources associated with the object.
DeviceCapabilities	Retrieves the capabilities of a printer device driver.
DrawEscape	Provides drawing capabilities of the specified video display that are not directly available through the graphics device interface.
EnumDisplayDevices	Retrieves information about the display devices in a system.
EnumDisplaySettings	Retrieves information about one of the graphics modes for a display device.
EnumDisplaySettingsEx	Retrieves information about one of the graphics modes for a display device.
EnumObjects	Enumerates the pens or brushes available for the specified device context.
EnumObjectsProc	An application-defined callback function used with the **EnumObjects** function.
GetCurrentObject	Retrieves a handle to an object of the specified type that has been selected into the specified device context.
GetDC	Retrieves a handle to a display device context for the client area of a specified window or for the entire screen.
GetDCBrushColor	Retrieves the current brush color for the specified device context.
GetObject	Retrieves information for the specified graphics object.
GetObjectType	Retrieves the type of the specified object.
GetStockObject	Retrieves a handle to one of the stock pens, brushes, fonts, or palettes.
ReleaseDC	Releases a device context, freeing it for use by other applications.
ResetDC	Updates the specified printer or plotter device context using the specified information.
SelectObject	Selects an object into the specified device context.
SetDCBrushColor	Sets the current device context brush color to the specified color value.
SetDCPenColor	Sets the current device context pen color to the specified color value.
SetLayout	Sets the layout for a device context.

Notes

Self Assessment

Fill in the blank:

9. ..... changes the settings of the default display device to the specified graphics mode.

14.6 Device Context Settings

14.6.1 Setting and Retrieving the Device Context Brush Color Value

The following example shows how an application can retrieve the current DC brush color by using the **SetDCBrushColor** and the **GetDCBrushColor** functions

Example:

```
SelectObject(hdc,GetStockObject(DC_BRUSH));
SetDCBrushColor(hdc,  RGB(00,0xff;00);
PatBlt(0,0,200,200,PATCOPY)
SetDCBrushColor(hdc,RGB(00,00,0xff);
PatBlt(0,0,200,200,PATCOPY);
```

14.6.2 Setting the Pen or Brush Color

The following example shows how an application can change the DC pen color by using the **GetStockObject** function or **SetDCPenColor** and the **SetDCBrushColor**functions.

Example:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
     int wmId, wmEvent;
     PAINTSTRUCT ps;
     HDC hdc;

     switch (message)
     {
     case WM_COMMAND:
          wmId     = LOWORD(wParam);
          wmEvent = HIWORD(wParam);
          // Parse the menu selections:
          switch (wmId)
          {
          case IDM_ABOUT:
                 DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
```

Notes

```
                    break;
                case IDM_EXIT:
                     DestroyWindow(hWnd);
                    break;
                default:
                       return DefWindowProc(hWnd, message, wParam, lParam);
               }
               break;
            case WM_PAINT:

               {
                     hdc = BeginPaint(hWnd, &ps);
                //     Initializing original object
                     HGDIOBJ original = NULL;

                //     Saving the original object
                       original = SelectObject(hdc,GetStockObject(DC_PEN));

                //     Rectangle function is defined as...
                //     BOOL Rectangle(hdc, xLeft, yTop, yRight, yBottom);

                //     Drawing a rectangle with just a black pen
                //     The black pen object is selected and sent to the current
                     device context
                //   The default brush is WHITE_BRUSH
                       SelectObject(hdc, GetStockObject(BLACK_PEN));
                      Rectangle(hdc,0,0,200,200);

                //     Select DC_PEN so you can change the color of the pen with
                //     COLORREF SetDCPenColor(HDC hdc, COLORREF color)
                       SelectObject(hdc, GetStockObject(DC_PEN));

                //     Select DC_BRUSH so you can change the brush color from the
                //     default WHITE_BRUSH to any other color
                       SelectObject(hdc, GetStockObject(DC_BRUSH));

               //     Set the DC Brush to Red
                //     The RGB macro is declared in "Windowsx.h"
                      SetDCBrushColor(hdc, RGB(255,0,0));

               //     Set the Pen to Blue
```

Notes

```
            SetDCPenColor(hdc, RGB(0,0,255));

        //     Drawing a rectangle with the current Device Context
            Rectangle(hdc,100,300,200,400);

        //     Changing the color of the brush to Green
            SetDCBrushColor(hdc, RGB(0,255,0));
            Rectangle(hdc,300,150,500,300);

        //     Restoring the original object
            SelectObject(hdc,original);

          // It is not necessary to call DeleteObject to delete stock
objects.
        }

        break;
     case WM_DESTROY:
          PostQuitMessage(0);
        break;
     default:
          return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

Self Assessment

Fill in the blank:

10. An application can retrieve the current DC brush color by using the **SetDCBrushColor** and the ..... functions.

14.7 Graphics Output

Graphics is a very important part of visual basic programming as an attractive interface will be appealing to the users. In the old BASIC, drawing and designing graphics are considered as difficult jobs, as they have to be programmed line by line in a text-based environment. However, in Visual Basic, these jobs have been made easy. There are four basic controls in VB that you can use to draw graphics on your form: the line control, the shape control, the image box and the picture box

14.7.1 Line and Shape Controls

To draw a straight line, just click on the line control and then use your mouse to draw the line on the form. After drawing the line, you can then change its color, width and style using the

Notes

BorderColor, BorderWidth and BorderStyle properties. Similarly, to draw a shape, just click on the shape control and draw the shape on the form.

Did u know? The default shape is a rectangle, with the shape property set at 0.

You can change the shape to square, oval, circle and rounded rectangle by changing the shape property's value to 1, 2, 3 4, and 5 respectively. In addition, you can change its background color using the BackColor property, its border style using the BorderStyle property, its border color using the BorderColor property as well its border width using the BorderWidth property.

14.7.2 The Image Box and the Picture Box

Using the line and shape controls to draw graphics will only enable you to create a simple design. In order to improve the look of the interface, you need to put in images and pictures of your own. Fortunately, there are two very powerful graphics tools you can use in Visual Basic which are the image box and the picture box.

To load a picture or image into an image box or a picture box, you can click on the picture property in the properties window and a dialog box will appear which will prompt the user to select a certain picture file. You can also load a picture at runtime by using the LoadPicture ( ) method. The syntax is

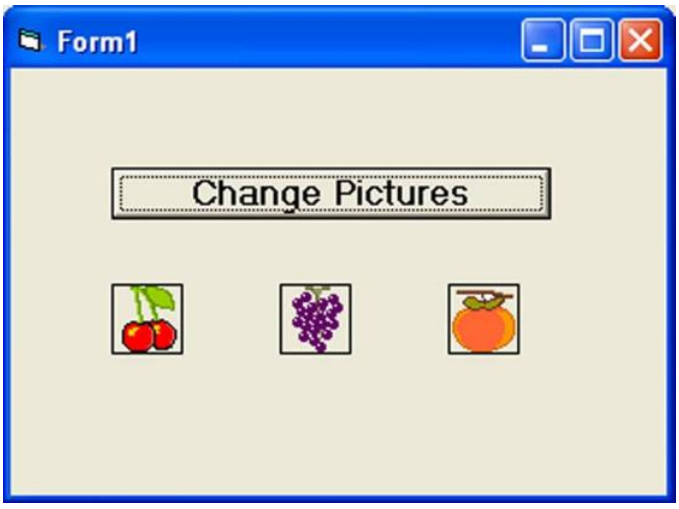
Image1.Picture= LoadPicture("C:\path name\picture file name") or

picture1.Picture= LoadPicture("C:\path name\picture name")

Example: The following statement will load the grape.gif picture into the image box.

Image1.Picture= LoadPicture("C:\My Folder\VB program\Images\grape.gif")

Example: In this example, each time you click on the 'change pictures' button as shown in Figure below, you will be able to see three images loaded into the image boxes. This program uses the Rnd function to generate random integers and then uses the LoadPicture method to load different pictures into the image boxes using the If…Then…Statements based on the random numbers generated. The output is shown in Figure below



Notes

```
Dim a, b, c As Integer
Private Sub Command1_Click ()
Randomize Timer
a = 3 + Int(Rnd * 3)
b = 3 + Int(Rnd * 3)
c = 3 + Int(Rnd * 3)

If a = 3 Then
Image1(0).Picture = LoadPicture("C:\My Folder\VB program\Images\grape.gif")
End If
If a = 4 Then
Image1(0).Picture = LoadPicture("C:\My Folder\VB program\Images\cherry.gif")
End If
If a = 5 Then
Image1(0).Picture = LoadPicture("C:\My Folder\VB program\Images\orange.gif")
End If
If b = 3 Then
Image1(1).Picture = LoadPicture("C:\My Folder\VB program\Images\grape.gif")
End If
If b = 4 Then
Image1(1).Picture = LoadPicture("C:\My Folder\VB program\Images\cherry.gif")
End If
If b = 5 Then
Image1(1).Picture = LoadPicture("C:\My Folder\VB program\Images\orange.gif")
End If
If c = 3 Then
Image1(2).Picture = LoadPicture("C:\My Folder\VB program\Images\grape.gif")
End If
If c = 4 Then
Image1(2).Picture = LoadPicture("C:\My Folder\VB program\Images\cherry.gif")
End If
If c = 5 Then
Image1(2).Picture = LoadPicture("C:\My Folder\VB program\Images\orange.gif")
End If
End Sub
```

Notes

Self Assessment

Fill in the blanks:

11. To draw a straight line, just click on the ..... control and then use your mouse to draw the line on the form.
12. The default shape is a rectangle, with the shape property set at .....

14.8 Animated Graphics

An animated GIF (Graphics Interchange Format) file is defined as a graphic image on a Web page that moves – for instance, a twirling icon or a banner with a hand that waves or letters that magically get bigger. Particularly, an animated GIF is a file in the Graphics Interchange Format mentioned as GIF89a that includes inside the single file a set of images that are exhibited in a particular order. An animated GIF can loop continuously (and it occurs as though your document never completes arriving) or it can exhibit one or a few sequences and then discontinue the animation. Animated GIFs are often utilized in Web ad banners.

One of the most exciting, fast-growing parts of the graphics world is animated graphics. With the growth of the Internet, computer-animated feature films and other new media, the animated graphics field has grown by leaps and bounds. Animated graphics are created with animation software. Much of the animated graphics on the Internet are created with Adobe Flash. Computer "3-D" animated feature films are created using high-end animation programs such as Autodesk Maya.

Adobe Flash

Adobe Flash has become the near-standard program for Web animation and is used as well for television and movie animation. The current version (as of 2009) of the program is called Adobe Flash CS4 Professional. It is useful for much more than animation, including many types of interactive Internet programming to create rich Internet experiences. It is possible to download Flash on a trial basis for free for a limited time. The program includes lessons and tutorials to help in learning.

Autodesk Maya

Autodesk Maya is a 3-D animation program allowing users to execute 3-D modeling, visual effects, animation and rendering. It is the industry standard for 3-D animators working in television, design, computer games and feature films. Like Flash, Maya is available in a trial version as a free download. There are several third-party plug-in applications available to enhance its productivity. It should also be noted that there are several other notable 3-D animation software products on the market, such as Kinetix 3D Studio Max.

Adobe After Effects

Another Adobe program in heavy use for animated graphics is Adobe After Effects. It is used to create special visual effects and animated graphics. Adobe After Effects is often used with other Adobe programs, such as Photoshop and Flash, as a postproduction tool to add special animation effects. Like other Adobe programs, it is available as a download on a free trial basis for a limited trial period.

Notes

Adobe Illustrator and Adobe Photoshop

Many animators use image editing programs such as Adobe Photoshop or vector-based programs such as Adobe Illustrator for drawing and coloring images for later use in animation software. Oliver Simonson of What Comics Entertainment uses both Photoshop and Illustrator for art and design work, then imports the images into Autodesk Maya for 3-D animation.

Careers in Animated Graphics

There are many specialized jobs in the animated graphics industry, such as modeler, animator, programmer, character designer, character animator, storyboarder and technical director, to name a few. While a general knowledge is useful, it is helpful to direct your training and education toward a specific job in the animated graphics industry. Many colleges and universities have created academic paths to study the field, but it is possible to self-train with animation software and find employment in the field. There are jobs at big companies like Pixar and Industrial Light and Magic, but the animated graphics industry is large and diversified into many areas beyond feature film effects, including advertising and Web animation.

Notes Java, Flash, and other tools can be utilized to attain the similar effects as an animated GIF. However, animated GIFs are usually simpler to create than comparable images with Java or Flash and generally slighter in size and therefore faster to display.

Self Assessment

Fill in the blank:

13. An ..... GIF (Graphics Interchange Format) file is defined as a graphic image on a Web page that moves.

14.9 The Peek Message [] Loop

To understand peek message(), firstly we will discuss GetMessage().

14.9.1 The GetMessage()

We use GetMessage() to create a loop that handled all the Windows message sent. However, there was a catch we didn't talk about at the time. Once we create the window, we get into the event loop, where we see the function GetMessage(). GetMessage() then waits for a message and, upon receiving one, sends it to the next step, TranslateMessage(). This is perfectly logical for Windows programming, because generally speaking Windows applications, Word for example, tend to sit and do nothing until you make a move.

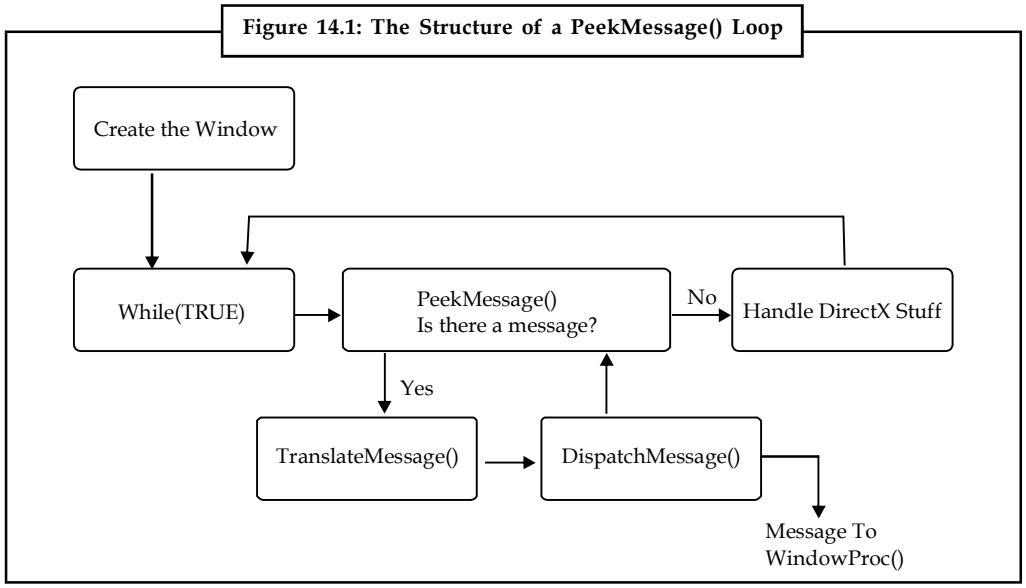
However, this doesn't work well for us. While all this waiting is going on, we need to be creating thirty to sixty fully-rendered 3D images per second and putting them on the screen without any delay at all. And so we are presented with a rather interesting problem, because Windows, if it sends any messages, will most definitely not be sending thirty of them per second.

Notes

14.9.2 A New Function, PeekMessage()

What we will do to solve this dilemma is replace our current GetMessage() function with a new function, PeekMessage(). This function does essentially the same thing, but with one important difference: it doesn't wait for anything. PeekMessage() just looks into the message queue and checks to see if any messages are waiting. If not, the program will continue on, allowing us to do what we need.

Figure 14.1: The Structure of a PeekMessage() Loop



Before we go any further, let's take a good look at PeekMessage(). Here is it's prototype.

```
BOOL  PeekMessage(LPMSG  lpMsg,
                    HWND  hWnd,
                    UINT  wMsgFilterMin,
                    UINT  wMsgFilterMax,
                    UINT  wRemoveMsg);
```

The first four parameters should be familiar to you. They are identical to the four parameters of GetMessage(). However, the fifth one, wRemoveMsg, is new.

What it does is indicate whether or not the message retrieved from the event queue should stay on the event queue or come off. We can put either PM_REMOVE or PM_NOREMOVE. The first one takes the messages off the queue when they are read, while the second one leaves the messages there for later retrieval. We will use the PM_REMOVE value here, and keep things simple.

So how do we implement this into our program? Following is the main loop from the last program we made, modified to use PeekMessage().

```
// Enter the infinite message loop
while(TRUE)
{
      // Check to see if any messages are waiting in the queue
       while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
```

Notes

```
    {
          // Translate the message and dispatch it to WindowProc()
          TranslateMessage(&msg);
          DispatchMessage(&msg);
    }
     // If the message is WM_QUIT, exit the while loop
     if(msg.message == WM_QUIT)
         break;
     // Run game code here
    // ...
    // ...
}
```

Now our program can handle things as timely as we please, without having to worry about Windows and its tedious messages.

Self Assessment

Fill in the blanks:

14. We use ..... to create a loop that handled all the Windows message sent.
15. ..... looks into the message queue and checks to see if any messages are waiting.

14.10 Summary

- In character mode, the display screen is treated as an array of blocks, each of which can hold one ASCII character.
- In graphics mode, the display screen is treated as an array of pixels, with characters and other shapes formed by turning on combinations of pixels.
- A device context is a Windows data structure containing information about the drawing attributes of a device such as a display or a printer.
- The **WM_PAINT** message is sent when the system or another application makes a request to paint a portion of an application's window.
- Graphics is a very important part of visual basic programming as an attractive interface will be appealing to the users.
- To draw a straight line, just click on the line control and then use your mouse to draw the line on the form.
- Similarly, to draw a shape, just click on the shape control and draw the shape on the form.
- An animated GIF (Graphics Interchange Format) file is defined as a graphic image on a Web page that moves.
- We use GetMessage() to create a loop that handled all the Windows message sent.
- PeekMessage() looks into the message queue and checks to see if any messages are waiting.

Notes

14.11 Keywords

Character Mode: In character mode, the display screen is treated as an array of blocks, each of which can hold one ASCII character.

Device Context: A device context is a Windows data structure containing information about the drawing attributes of a device such as a display or a printer.

Graphics mode: In graphics mode, the display screen is treated as an array of pixels, with characters and other shapes formed by turning on combinations of pixels.

PeekMessage(): PeekMessage() looks into the message queue and checks to see if any messages are waiting.

WM_PAINT: The **WM_PAINT** message is sent when the system or another application makes a request to paint a portion of an application's window.

14.12 Review Questions

1. Make distinction between character mode and graphic mode.
2. What is a device context? Illustrate the functions of device context.
3. Illustrate the use of **WM_PAINT** message.
4. Discuss the functions that are used for changing device contexts.
5. Illustrate with example the concept of Setting the Pen or Brush Color.
6. What is graphics output? Elucidate the basic controls in VB that you can use to draw graphics on your form.
7. Make distinction between GetMessage() and PeekMessage().
8. Explicate the Structure of a PeekMessage() Loop.
9. Elucidate the use of CPaintDC objects and CClientDC objects.
10. You can load a picture at runtime by using the LoadPicture ( ) method. Comment.

Answers: Self Assessment

1.	character-based	2.	pixels
3.	device context	4.	CPaintDC
5.	drop_caps	6.	drop_caps
7.	WM_PAINT	8.	DispatchMessage
9.	ChangeDisplaySettings	10.	GetDCBrushColor
11.	line	12.	0
13.	animated	14.	GetMessage()
15.	PeekMessage()		

14.13 Further Readings

Notes

Books

Brent E. Rector, *Win32 Programming*, Addison-Wesley

Charles Petzold, *Programming Windows*, Charles Petzold

Roger Mayne, *Windows and Graphics Programming with Visual C++.NET*, World Scientific

Online link

www.stat.auckland.ac.nz/~paul/RGraphics/chapter1.pdf