Introduction to Windows Programming

Windows

- Windows is GUI operating systems developed by Microsoft.
- Each version of Windows includes a graphical user interface, with a desktop that allows users to view files and folders in windows.
- Windows has been the most widely used operating system for personal computers (PCs).

Versions of Windows or History of Windows:

Microsoft Windows is designed for both home computing and professional purposes.

Past versions of Windows home editions include :

Windows 3.0 (1990),

Windows 3.1 (1992),

Windows 95 (1995),

Windows 98 (1998),

Windows Me (2000),

Windows XP (2001),

Windows Vista (2006).

Windows 7 (2008).

Windows 8 (2010)

The current version is Windows 10 ,released in 2013

Windows System 32

- The Windows **System32** directory is often located in either C:\Windows\System32 or C:\Winnt\system32. Often, many Microsoft Windows error messages will contain the system32 directory because many of the system files Windows uses to run are stored in this directory.

WIN API

- The Windows application programming interface (API) lets you develop desktop and server applications that run successfully on all versions of Windows.
- It is taking advantage of the features and capabilities unique to each version.
- The Windows API can be used in all Windows-based desktop applications, and the same functions are generally supported on 32-bit and 64-bit Windows.

Windows Fundamentals

- Windows applications can be developed using a procedure-oriented approach in either C or C++.
- All approaches bring together point-and-shoot control, pop-up menus, and the ability to run applications written especially for the Windows environment.
- Windows gives the ability to develop graphics user interface(GUI)

The Windows Environment

- Windows is a graphics-based multitasking operating system.
- Programs developed for this environment have a consistent look and command structure.
- To the user, this makes learning each successive Windows application easier.
- To help in the development of Windows applications, Windows provides numerous **built-in** functions that allow for easy implementation of menus, scroll bars, dialog boxes, icons that represent a user-friendly interface.
- Windows permits the application to work in a hardware-independent manner.

The Windows Environment

Install the Windows SDK

To write a Windows program in C or C++, you must install the Microsoft Windows Software Development Kit (SDK) or a development environment that includes the Windows SDK, such as Microsoft Visual C++. The Windows SDK Contains the headers and libraries necessary to compile and link your application. The Windows SDK also contains command-line tools for building Windows applications, including the Visual C++ compiler and linker. Although you can compile and build Windows programs with the command-line tools, we recommend using a full-featured development environment such as Microsoft Visual Studio. Visual C++ Express is a free downloadable edition of Visual C++ available at :

http://go.microsoft.com/fwlink/?LinkId=181514.

Microsoft
Visual C++ 6.0
Enterprise Edition
Microsoft
VISUAL
This program is protected by U.S. and international copyright laws as described in Help About.

Prepare Your Development Environment

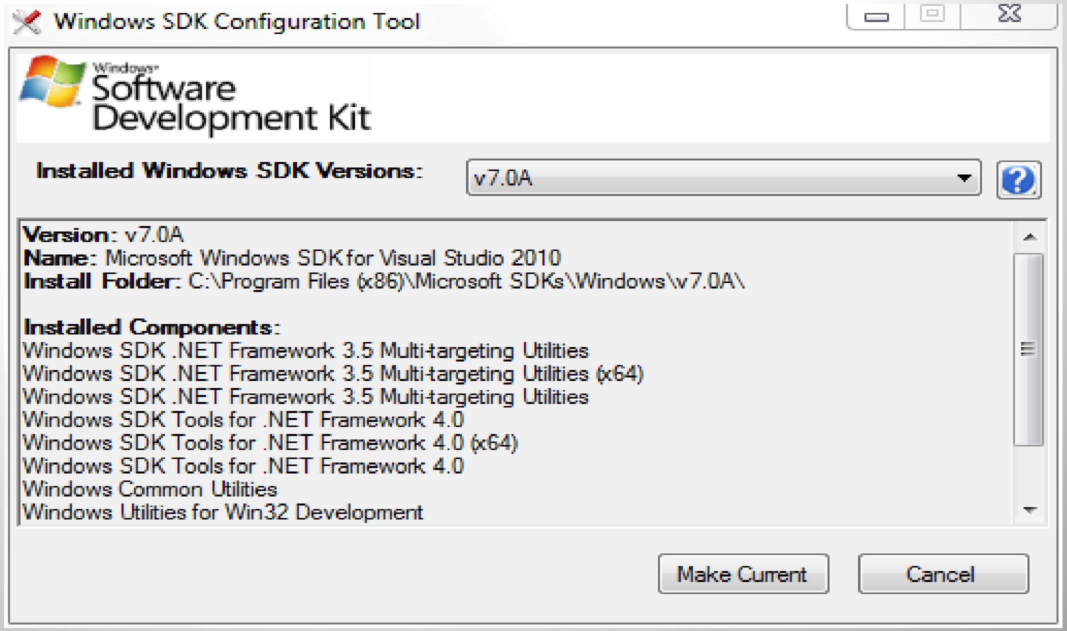
Install the Windows SDK

- The Windows SDK supports development of both 32-bit and 64-bit applications. In fact, the Windows APIs are designed so that the same code can compile for 32-bit or 64-bit without changes.

Set Include and Library Paths

- After you install the Windows SDK, make sure that your development environment points to the Include and Lib folders, which contain the header and library files.
- For Visual Studio, the Windows SDK includes a Visual Studio Configuration Tool. This tool updates Visual Studio to use the Windows SDK header and library paths.

A screen shot of the Visual Studio Configuration Tool


Windows SDK Configuration Tool
Windows Software Development Kit
Make Current
Cancel

Installed Windows SDK Versions: v7.0A

```
Version: v7.0A
Name: Microsoft Windows SDK for Visual Studio 2010
Install Folder: C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\

Installed Components:
Windows SDK .NET Framework 3.5 Multi-targeting Utilities
Windows SDK .NET Framework 3.5 Multi-targeting Utilities (x64)
Windows SDK .NET Framework 3.5 Multi-targeting Utilities
Windows SDK Tools for .NET Framework 4.0
Windows SDK Tools for .NET Framework 4.0 (x64)
Windows SDK Tools for .NET Framework 4.0
Windows Common Utilities
Windows Utilities for Win32 Development
```

Windows Advantages

Graphics User Interface (GUI)

- ➢ All versions of Windows are based on the same standardized interface.
- ➢ This interface uses pictures, or icons, to represent disk drives, files, subdirectories, and many of the operating system commands and actions.

- **Multitasking Environment**

- ➢ The Windows multitasking environment allows the user to have several applications, or several instances of the different application, running at the same time.
- ➢ Each application occupies a rectangular window on the screen.

Windows Advantages(contd.)

- **Advantages of Using a Queued (serial wise) Input**

➢ Windows receives all input from the keyboard, mouse, and timer in the system queue.

➢ It is the queue's responsibility to redirect the input to the appropriate program since more than one application can be running.

➢ It is achieved by copying the message from the system queue into the application's queue.

➢ When application is ready to process the input, it reads from its queue and dispatches a message to the correct window.

➢ Input is accessed with the use of a uniform format called an ***input message.***

Windows Advantages(contd.)

OOPs and Windows Messages

- Windows has always employed a OOP environment.
- Message is a notification that some event of interest has occurred that may or may not need a specific action.
- User may initiate these events by clicking or moving the mouse, changing the size of a window, or making a menu selection.
- The events can also be initiated by the application itself e.g. a graphics-based spreadsheet could finish a recalculation that results in the need to update a graphics bar chart.
- It is the message system that allows Windows to achieve its multitasking capabilities.
- The message system makes it possible for Windows to share the processor among different applications.

Windows Advantages(contd.)

- **Managing Memory**
- ➢ Most important shared resources under Windows is system memory.
- ➢ As new programs are started and old ones are terminated, memory become fragmented.
- ➢ Windows is capable for free memory space by moving blocks of code and data in memory.
- **Dynamic Link Libraries (DLL)**
- ➢ Supports much of Windows' functionality.
- ➢ Enhance the base operating system by providing a powerful GUI.

Dynamic Linking:

Central to the workings of Windows is a concept known as "dynamic linking." Windows provides a wealth of function calls that an application can take advantage of, mostly to implement its user interface and display text and graphics on the video display. These functions are implemented in dynamic−link libraries, or DLLs. These are files with the extension .DLL or sometimes .EXE, and they are mostly located in the \WINDOWS\SYSTEM subdirectory under Windows 98 and the \WINNT\SYSTEM and \WINNT\SYSTEM32 subdirectories under Windows NT.

Layout of a Window

- **Border**
- **Title Bar**
- **Control Icon**
- **System Menu**
- **Minimize Icon**
- **Maximize Icon**
- **Close Window Icon**
- **Vertical Scroll Bar,** if desired.
- **Horizontal Scroll Bar,** if desired.
- **Menu Bar(optional)**

Windows Graphics Objects

- **Menus**
- **Title bars**
- **Control boxes**
- **Scroll bars**
- **Icons**
- **Cursors**
- **Message Boxes**
- **Windows Dialog Boxes**
- **Fonts**
- **Pens**
- **Brushes**

How Windows' Applications Handled

- Windows provides an application program with access to hundreds of function calls, directly or indirectly, through foundation classes.
- These function calls are handled by several main modules-
  - KERNEL- responsible for memory management, , loading and running an application, and scheduling.
  - GDI (graphics device interface)- contains all of the routines to create and display graphics
  - USER modules- takes care of all other application requirements.

The Windows Message Format

- Messages are used to notify a program that an event of interest has occurred.
- Only one message system exists under Windows—the system message queue.
- Each program currently running under Windows also has its own program message queue.
- The USER module must transfer each message in the system message queue to a program’s message queue.
- The program’s message queue stores all messages for all windows in that program.

Frequently Used Win32 Data Types

CALLBACK	Replaces FOR PASCAL in application's call back routine.
HANDLE	32-bit unsigned integer that is used as a handle.
HDC	Handle to a device context.
HWND	32-bit unsigned integer that is used as the handle to a window.
LONG	32-bit signed integer
LPARAM	Type used for declaration of lParam(Long pointer).
LPCSTR	LPCSTR is the same as LPSTR, but is used for read-only string pointers.
LPSTR	32-bit pointer.
LPVOID	A generic pointer type. It is equivalent to (void *).
LRESULT	Used for the return value of a window procedure.
UINT	An unsigned integer type
WCHAR	A 16-bit UNICODE character. WCHAR is used to represent all of the symbols for all of the world's languages.
WINAPI	Replaces FAR PASCAL in API declarations.
WPARAM	Used for the declaration of wParam.
HINSTANCE	Handle to the current instance

Frequently Used Win32 Structures

MSG	Defines the fields of an input message
PAINTSTRUCT	Defines the paint structure used when drawing inside a window
RECT	Defines a rectangle
WNDCLASS	Defines a window class

Handles

- Used when writing procedure-oriented Windows applications.
- Handle is a unique number that identifies objects like-
- Windows
- Controls
- Menus
- Icons
- Pens
- Brushes
- Memory allocation
- Output devices
- Window instances-Each copy of a program loaded in main memory is called an instance.

Instance Handles

- Windows allows to run more than one copy of the same application at the same time, so operating system needs to keep track of each of these instances.
- It does this by attaching a unique instance handle to each running copy of the application.

Windows Header File: WINDOWS.H

- Provides a path to over a thousand constant declarations, **typedef** declarations, and hundreds of function prototypes
- Main reasons a Windows application takes longer to compile than a non-Windows C or C++ program is the size of this and associated header files.
- Traditionally, WINDOWS.H is a required **include** file in all C and C++ Windows applications.

Calling Convention for Functions

- The parameters for the function are pushed from the rightmost parameter to the leftmost parameter, in a normal C and C++ fashion.
- Function declarations under 16-bit Windows 3.x included the **PASCAL** modifier, which was more efficient under DOS in which parameters are pushed onto the stack from left to right.
- Windows does not use this modifier for 32-bit applications and instead of it uses _stdcall.
- PASCAL’s use in 32-bit windows application will not give error, just only warning that _stdcall is not used.

Windows Application Components

Windows applications contain two common and essential elements-

1. WinMain( ) function
2. Window function.

WinMain() function:

- ➢ WinMain( ) serves as the entry point for the Windows application
- ➢ Acts in a way similar to the main( ) function in standard C or C++ programs.
- ➢ Responsible for the following:
- ▪ Creating and initiating application’s message processing loop
- ▪ Performing any required initializations
- ▪ Registering the application’s **window** class
- ▪ Terminating the program

WinMain() (contd.)

- Four parameters are passed to the WinMain( ) function from Windows.
- WinMain function is defined as-

  int _stdcall WinMain(HINSTANCE hInstance , HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)

hInstance contains the instance handle of the application. This number uniquely identifies the program when it is running under Windows.

hPrevInstance will always contain a NULL indicating that there is no previous instance of this application.

MS-DOS versions of Windows (Windows 3.3 and earlier) used hPrevInstance to indicate whether there were any previous copies of the program loaded. Under operating systems, such as Windows 95, 98, and NT, each application runs in its own separate address space. For this reason, under Windows 95, 98, and NT, it returns just NULL.

lpszCmdLine is a long pointer to a null-terminated string that represents the application's command-line arguments. Normally, lpszCmdLine contains a NULL if the application was started using the Windows Run command.

nCmdShow defines the possible ways a window can be displayed, such as SW_SHOWNORMAL, SW_SHOWMAXIMIZED, or SW_MINIMIZED.

WNDCLASS

- **Window** class serves as a template to defines attributes of combination of user-selected styles, fonts, caption bars, icons, size, and so on.
- WinMain( ) function registers the application's main **window** class.
- Same standard C and C++ structure type is used for all Windows class definitions.
- Predefined **window** classes are available, but most programmers define their own **window** class.

An example WNDCLASS

- Following example is taken directly from WINUSER.H, which is an **#include** file referenced in WINDOWS.H.
- The header file contains a **typedef** statement defining the structure type WNDCLASSW (a UNICODE-compatible definition), from which WNDCLASS is derived:

```
  typedef struct tagWNDCLASSW {
   UINT        style;            //
   WNDPROC     lpfnWndProc;
   int         cbClsExtra;
   int         cbWndExtra;
   HANDLE      hInstance;
   HICON       hIcon;
   HCURSOR     hCursor;
   HBRUSH      hbrBackground;
   LPCWSTR     lpszMenuName;
   LPCWSTR     lpszClassName;
} WNDCLASSW, *PWNDCLASSW, NEAR *NPWNDCLASSW, FAR
*LPWNDCLASSW;
```

Fields of WNDCLASS

- Style

The style field names the **class** style. The styles can be combined with the bitwise OR operator.

Frequently Used Windows Styles

CS_HREDRAW	Redraws the window when horizontal size changes.
CS_VREDRAW	Redraws the window when the vertical size changes
CS_GLOBALCLASS	States that the window class is an application
CS_NOCLOSE	Inhibits the close option from the system menu
CS_SAVEBITS	Saves that part of a screen that is covered by another window
CS_CLASSDC	Provides the window class a display context
CS_SAVEBITS	Saves that part of a screen that is covered by another window

Fields of WNDCLASS (contd)

lpfnWndProc

- Receives a pointer to the window function that will carry out all of the tasks for the window.

cbClsExtra

- Gives the number of bytes that must be allocated after the **window** class structure. It can be set to NULL. The number of extra bytes to allocate following the window-class structure. The system initializes the bytes to zero.

cbWndExtra

- Gives the number of bytes that must be allocated after the window instance. It can be set to NULL.

The number of extra bytes to allocate following the window instance. The system initializes the bytes to zero.

hInstance

- Defines the instance handle of the application registering the **window** class. This cannot be set to NULL.

hIcon

- Icon to be used when the window is minimized. This can be set to NULL.

hCursor

- the cursor to be used with the application. This handle can be set to NULL. The cursor is valid only within the application’s client area.

Fields of WNDCLASS (contd)

hbrBackground

- Identification for the background brush. This can be a handle to the physical brush or it can be a color value. Color values must be standard colors such as-

  COLOR_ACTIVEBORDER
  COLOR_ACTIVECAPTION

  COLOR_WINDOW
  COLOR_WINDOWFRAME
  COLOR_WINDOWTEXT

  COLOR_MENU
  COLOR_MENUTEXT
  COLOR_SCROLLBAR
- If hbrBackground is set to NULL, the application paints its own background.

Fields of WNDCLASS (contd)

lpszMenuName

- Pointer to a null-terminated character string.
- The string is the resource name of the menu.
- This item can be set to NULL.

lpszClassName

- Pointer to a null-terminated character string.
- The string is the name of the **window** class.

Defining a Window Class

- Applications can define WNDCLASS by filling the structure’s fields with the information about the window class.

```
WNDCLASS wndclass;
wndclass.lpszClassName=szProgName;
wndclass.hInstance    =hInstance;
wndclass.lpfnWndProc  =WndProc;
wndclass.hCursor      =LoadCursor(NULL,IDC_ARROW);
wndclass.hIcon        =NULL;
wndclass.lpszMenuName =szApplName;
wndclass.hbrBackground=GetStockObject(WHITE_BRUSH);
wndclass.style        =CS_HREDRAW|CS_VREDRAW;
wndclass.cbClsExtra   =0;
wndclass.cbWndExtra   =0;
if (!RegisterClass (&wndclass))
 return 0;
```

Creating a Window

- Window class defines the general characteristics of a window, allowing the same window class to be used for many different windows.
- While the parameters for CreateWindow( ) specify more detailed information about the window.
- Returns the handle of the newly created window. Otherwise, the function returns a NULL value.
- Parameter information falls under the following categories:
  - the class,
  - title,
  - style,
  - screen position,
  - window’s parent handle,
  - menu handle,
  - instance handle,
  - 32 bits of additional information

Windows Data Types

The data types supported by Windows are used to define function return values, function and message parameters, and structure members. They define the size and meaning of these elements.

Data Type Ranges

Visual C++ 32-bit and 64-bit compilers recognize the types in the table later in this article.

- **int** (**unsigned int**)
- **__int8** (**unsigned __int8**)
- **__int16** (**unsigned __int16**)
- **__int32** (**unsigned __int32**)
- **__int64** (**unsigned __int64**)
- **short** (**unsigned short**)
- **long** (**unsigned long**)
- **long long** (**unsigned long long**)

If its name begins with two underscores (**__**), a data type is non-standard.

The ranges that are specified in the following table are inclusive-inclusive.

THANK YOU