Windows Programming CSA2040

Kristian Guillaumier
http://www.cs.um.edu.mt/~kguil
kguil@cs.um.edu.mt

Getting Started

- Examples are in C and/or PowerBASIC. It will be trivial to port from and to different languages.
- All the examples in the "Petzold" book are available ported to PowerBASIC. You can get them from: http://www.powerbasic.com/files/pub/pbwin/Petzold.zip
- You can download the Borland C/C++ Compiler Version 5.5 for free from: http://www.borland.com/products/downloads/download_cbuilder.html
- There is a good WIN32 tutorial at: http://www.winprog.org/tutorial/ - a number of examples here are borrowed from this site.
- www.allapi.net is cool.
- msdn.microsoft.com is the definitive resource.

Recommended Books

- **Programming Windows, The Definitive Guide to the Win32 API** by Charles Petzold, 5th edition, Microsoft Press, ISBN: 157231995X.
- **Windows Programming with C++** by Henning Hansen, Addison Wesley Professional, ISBN: 0201758814.

Getting Started

- Programming Windows, requires you to understand the services offered by the WIN32 Application Programming Interface (API).
- The API consists of a number of DLLs containing common Windows functions.
  - Kernel32.dll
  - GDI32.dll
  - User32.dll
  - …
- To access the API functions, constant declarations and types you will need to “wrap” them in your code. In C this is already available in “windows.h” and in PowerBASIC, this is available in “WIN32API.INC”.

WIN32 Hello World

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR     lpCmdLine,
                   int       nCmdShow)
{
  MessageBox(NULL, "Hello World!",
                   "My Caption", MB_OK);
  return 0;
}
```


What's Going On? (1)

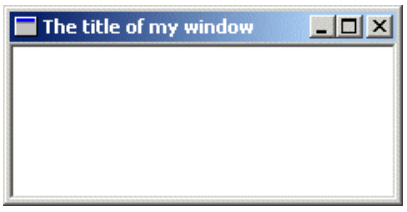
- **windows.h** contains the declarations of functions and constants such as *MessageBox*, *MB_OK*, *HINSTANCE* and *LPSTR*.
- **WinMain** is the equivalent of the main() functions in C – it is the starting point of a *Windows* application:
  - **hInstance**: Handle/pointer to the EXE in memory.
  - **hPrevInstance**: Always NULL – Never used.
  - **lpCmdLine**: Pointer to the command line string.
  - **nCmdShow**: an integer determining whether the window will be visible/hidden/…

What's Going On? (2)

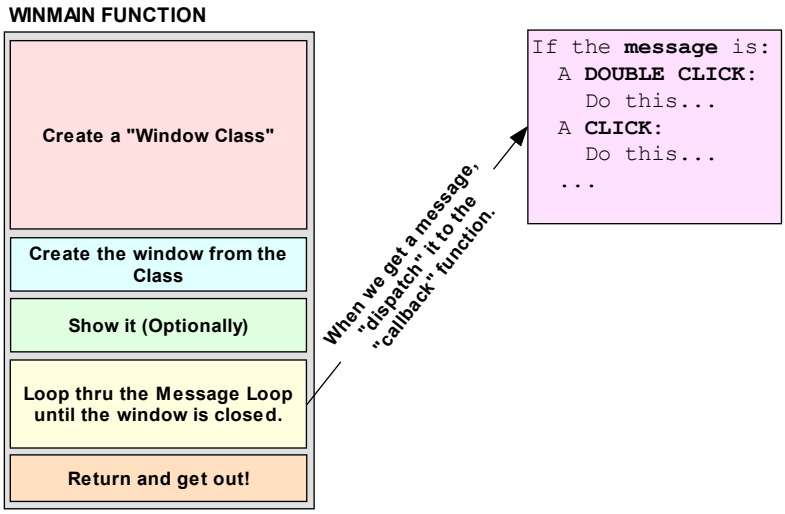
- The *hInstance* handle is used as the pointer to the EXE in memory so it is useful to locate resources such as images and icons in the program.
- In C, the **WINAPI** calling convention before the *WinMain* function is equivalent to *_stdcall*. In some languages such as PB it is not necessary.
- The WIN32 header file defines a number of types such as **LPSTR** (this is exactly equivalent to char*).

More Windows

- We will now see how to create a simple window like:



General Structure (1)



The Window Class

- **A Window Class is NOT related to Object Oriented Software.**
- In Windows everything is more or less a Window (including a button, combo box, …). The type of window is determined by it's class.
- A Window Class is a special structure (**WNDCLASS**) that is populated to specify the general properties of the window (e.g. it's icon, background colour, cursor, …).
- One of the most important properties is the definition of the Callback function (more on this later).
- Once a class has been created it is "Registered".

Creating the Window

- Once a class structure has been populated and registered, a window is created based on it.
- To create the window, the **CreateWindow** function is used (There is a variant called CreateWindowEx).
- Some arguments, CreateWindow takes are:
  - The class name to base this window on.
  - The text in the title bar.
  - The type of window (e.g. borderless, tool window, …)
  - The x,y coordinate of the top-left corner – in pixels.
  - The width and height of the window – in pixels.
  - …

Showing the Window (1)

- The CreateWindow function returns a handle/pointer to the window just created. The window is not yet visible.
- The window will be referred to it using its handle.
- To show the window, the **ShowWindow** API call is used. ShowWindow takes 2 arguments:
  - The handle of the window to show/hide.
  - An integer constant determining whether to show/hide the window.

Showing the Window (2)

- The show command can be:
  - **SW_HIDE** - Hides the window and activates another window.
  - **SW_MAXIMIZE** - Maximizes the specified window.
  - **SW_MINIMIZE** - Minimizes the specified window and activates the next top-level window in the Z order.
  - **SW_RESTORE** - Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.
  - **SW_SHOW** - Activates the window and displays it in its current size and position.
  - …
- Usually after a call to ShowWindow, another call to **UpdateWindow** is made. This call basically makes sure the window is displayed correctly.

The Message Loop (1)

- A windows program has a special "**message queue**".
- Whenever something happens to the window a message is placed in its queue. For example, if the window is clicked a "click" message is placed on its queue.
- We will use a while loop to retrieve messages from this queue and send them to the callback function for processing.
- Each message is defined by an integer constant. For example the WM_CREATE message is sent to the window when it is created. It is more-or-less equivalent to the Form_Load event in Visual Basic.

The Message Loop (2)

- Each message can be accompanied by some parameters. For example a mouse move message would be accompanied by the corresponding x and y mouse coordinates.
- These parameters are sent together with the message to the callback function.
- You can have a maximum of 2 parameters. These are called wParam and lParam. They are both long integers (signed 32-bit).
- Note that wParam and lParam being long integers may be pointers to whole data structures.

The Real Thing (1)

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
                                                                    int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;

    //Step 1: Registering the Window Class
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = 0;
    wc.lpfnWndProc = WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

    if(!RegisterClassEx(&wc))
    {
      MessageBox(NULL, "Window Registration Failed!", "Error!", MB_ICONEXCLAMATION | MB_OK);
      return 0;
    }
```

Prepare the Window Class

Tell the Class which function will act as the Callback

Register the Class

The Real Thing (2)

```
    // Step 2: Creating the Window
    hwnd = CreateWindowEx(WS_EX_CLIENTEDGE,
                          g_szClassName,
                          "The title of my window",
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT,
                          CW_USEDEFAULT,
                          240, 120, NULL, NULL,
                          hInstance, NULL);

    if (hwnd == NULL)
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
                          MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }
    ShowWindow(hwnd, nCmdShow);

    UpdateWindow(hwnd);

    // Step 3: The Message Loop
    while(GetMessage(&Msg, NULL, 0, 0) > 0)
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```

Create the Window

Show It

Loop Thru the Messages the Window Receives and send them to the callback function,

The Real Thing (3)

```
// Step 4: the Window Procedure
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch (msg)
    {
        case WM_CLOSE:
            DestroyWindow(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```

Select which Message **WE CHOOSE** to Handle

Those we do not handle ourselves, we'll ask Windows to process!