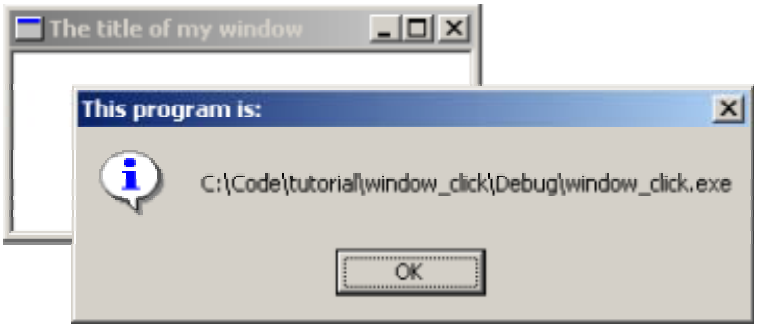
LECTURE-3

HANDLING MESSAGES

Example: window_click

In lecture two we have written a program for a simple window. Although window is created but it doesn't do anything except what DefWindowProc() allows it to, like be sized, maximised, etc... Not really all that exciting.



In the next section we are going to show how to modify what you already have to do something new.

Okay for starters take the example code for the last window we worked on and make sure it compiles and runs as expected. Then you can either keep working on it for the next little bit or copy it to a new project to modify.

We're going to add the capability to show the user what the name of our program is when they click on our window. Not very exciting, it's basically to get the hang of handling messages. Lets look at what we have in our **WndProc():**

```
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;
```

```
}
```

If we want to handle mouse clicks, we need to add a `WM_LBUTTONDOWN` handler (or `WM_RBUTTONDOWN`, `WM_MBUTTONDOWN`, for right and middle clicks respectively).

If we refers to *handling a message* this means to add it into the `WndProc()` of your window class as follows:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
   switch(msg)
   {
      case WM_LBUTTONDOWN:    // <-
                              // <-     we just added this stuff
      break;                  // <-
      case WM_CLOSE:
         DestroyWindow(hwnd);
      break;
      case WM_DESTROY:
         PostQuitMessage(0);
      break;
      default:
         return DefWindowProc(hwnd, msg, wParam, lParam);
   }
   return 0;
}
```

The order in which you handle your messages rarely matters. Just make sure you've got your `break;` after each one. As you can see we added another `case` into our `switch()`. Now we want something to happen when we get to this part of our program.

First we will present the code we want to add (that will show the user the filename of our program) and then we will integrate it into our program. Later on we will probably just show you the code and let you integrate it into your program. This is of course better for us as we don't have to type as much and it's better for you because you will be able to add the code into ANY program and not just the ones is presented here.

```
GetModuleFileName(hInstance, szFileName, MAX_PATH);
MessageBox(hwnd, szFileName, "The program by VP Singh:", MB_OK |
MB_ICONINFORMATION);
```

Now this code does not stand on it's own, it can't just be slapped into our code any old place. We specifically want it to run when the user clicks the mouse button so this is how we would merge this small bit of code into our skeleton program:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
    switch(msg)
    {
        case WM_LBUTTONDOWN:
// BEGIN NEW CODE
        {
            char szFileName[MAX_PATH];
```

```
            HINSTANCE hInstance = GetModuleHandle(NULL);

            GetModuleFileName(hInstance, szFileName, MAX_PATH);
            MessageBox(hwnd, szFileName, "The program by Dr. VP
Singh:", MB_OK | MB_ICONINFORMATION);
        }
// END NEW CODE
        break;
        case WM_CLOSE:
            DestroyWindow(hwnd);
        break;
        case WM_DESTROY:
            PostQuitMessage(0);
        break;
        default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```

Note the new set of curly braces {} . These are required when declaring variables inside a switch() statement.

So if you've added in that code, compile it now. If it works, click on the window and you should see a box with the name and path of the program file.

You'll notice we've added two variables, **hInstance** and **szFileName**. Look up **GetModuleFileName()** and you will see that the first parameter is a **HINSTANCE** referring to the executable module (our program, the .exe file). Where do we get such a thing? **GetModuleHandle()** is the answer. The references for GetModuleHandle() indicate that passing in NULL will return us "*a handle to the file used to create the calling process*", which is exactly what we need, the HINSTANCE just mentioned. Putting all this information together we end up with the following declaration:

```
HINSTANCE hInstance = GetModuleHandle(NULL);
```

Now on to the second parameter, again turning to our trusty reference manual, we see that it is " *a pointer to a buffer that receives the path and file name of the specified module*" and the data type is LPTSTR (or LPSTR if your references are old). Since LPSTR is equivalent to char* we can declare an array of char's like this:

```
char szFileName[MAX_PATH];
```

MAX_PATH is a handy macro included via <windows.h> that is defined to the maximum length of a buffer needed to store a filename under Win32. We also pass MAX_PATH to GetModuleFileName() so it knows the size of the buffer.

After GetModuleFileName() is called, the buffer szFileName will be filled with a null terminated string containing the name of our .exe file. We pass this value to MessageBox() as an easy way of displaying it to the user.

So if you've added in that code, compile it now. If it works, click on the window and you should see a box with the name of the .exe pop up.

If it doesn't work, here's the full code to the program. Compare it to what you have and see what, if any, mistakes you made.

WNDCLASSEX structure

Every window that you create has an associated WNDCLASSEX structure. The WNDCLASSEX structure provides all the information necessary for Windows(tm) to do perform window related functions like drawing its icon, cursor, menu, calling the callback function which will receive messages and so on.

The WNDCLASSEX structure is as follows:

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

cbSize

This must always be set to sizeof(WNDCLASSEX).

style

This specifies the class styles. Take a look at your SDK documentation for the values this member can take.

lpfnWndProc

Pointer to the WndProc which will handle this windows' messages.

cbClsExtra

Number of extra bytes to allocate at the end of the WNDCLASSEX structure.

cbWndExtra

Number of extra bytes to allocate at the end of the window instance.

hInstance

Identifies the instance that the window procedure of this class is within.

hIcon

Handle to the icon associated with windows of this class.

hCursor

Handle to the cursor for windows of this class.

hbrBackground

Identifies the class background brush.

lpszMenuName

Identifies the menu for windows of this class.

lpszClassName

Pointer to a NULL terminated string or an atom specifying the class of this structure.

hIconSm

Handle to the small icon associated with this class.

Program to create an interactive window

```
#include <windows.h>

const char g_szClassName[] = "myWindowClass";

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
    switch(msg)
```

```
    {
        case WM_LBUTTONDOWN:
        {
            char szFileName[MAX_PATH];
            HINSTANCE hInstance = GetModuleHandle(NULL);

            GetModuleFileName(hInstance, szFileName, MAX_PATH);
            MessageBox(hwnd, szFileName, "This program is:", MB_OK |
MB_ICONINFORMATION);
        }
        break;
        case WM_CLOSE:
            DestroyWindow(hwnd);
        break;
        case WM_DESTROY:
            PostQuitMessage(0);
        break;
        default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}

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

    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;
    }

    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);

    while(GetMessage(&Msg, NULL, 0, 0) > 0)
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```