C++, VC6, VC7, Windows, Win95, Win98, WinME, NT4, Win2K, WinXP, VS, VS6, Dev

Posted : **10 Jul 2002**
Updated : **11 Jul 2002**
Views : **86,789**

A Simple Win32 Window Wrapper Class

By Jason Henderson

How to make a Win32 application object oriented, without using MFC.

- Download source files - 5.8 Kb
- Download demo project - 11.5 Kb

Introduction

I like object-oriented programming. In my professional C++ career, I have used nothing but MFC. So, when I want to make a good ole' fashioned Win32/SDK application, I cringe at the thought of not using the OO paradigm. If you're like me and you want to get away from the old C programming style, here is some code that will help you write your own window wrapper class.

Where are all of the examples?

It's nearly impossible to find good examples of a window wrapper class. None of my game development books have one. None of my general Windows programming books do either. Conducting a Google search only gave me one good example, Creating a Win32 Window Wrapper Class. You would think more people would be doing this. Unfortunately, there aren't more examples because its not easy to get it working.

The main problem

When you create a `WNDCLASSEX` structure you need to give the structure a pointer to your window procedure in the `lpfnWndProc` member. The window procedure is a callback, a special kind of function called directly by Windows when needed.

To make callbacks work correctly, strict typedefs are defined in the Windows header files. If you try to use a class member as a callback, the compiler will give you an error, saying that the member prototype does not match the required typedef.

The solution

To get around this compiler error, we need to make the window procedure `static`. From this static method we then call another method of the class to actually handle the messages.

To call the message handler of the correct window, we need to get the pointer of the window class using the `SetWindowLong` and `GetWindowLong` API functions. We send the `this` pointer to the window procedure via the `CreateWindow` function's `LPVOID lpParam // window-creation data` parameter.

When the `WM_NCCREATE` message is received, the window procedure sets the window long value to the pointer and is then able to call the correct message handler.

```
class CBaseWindow
{
public:
    // there are more members and methods

    // but you can look at the code to see them


    // static message handler to put in WNDCLASSEX structure

    static LRESULT CALLBACK stWinMsgHandler(HWND hwnd,
                UINT uMsg, WPARAM wParam, LPARAM lParam);
protected:
    // the real message handler

    virtual LRESULT CALLBACK WinMsgHandler(HWND hwnd,
                UINT uMsg, WPARAM wParam, LPARAM lParam)=0;

    // returns a pointer the window (stored as the WindowLong)

    inline static CBaseWindow *GetObjectFromWindow(HWND hWnd)
    {
        return (CBaseWindow *)GetWindowLong(hWnd, GWL_USERDATA);
    }
};

LRESULT CALLBACK CBaseWindow::stWinMsgHandler(HWND hwnd,
                    UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    CBaseWindow* pWnd;

    if (uMsg == WM_NCCREATE)
    {
        // get the pointer to the window from

        // lpCreateParams which was set in CreateWindow
        SetWindowLong(hwnd, GWL_USERDATA,
           (long)((LPCREATESTRUCT(lParam))->lpCreateParams));
    }

    // get the pointer to the window

    pWnd = GetObjectFromWindow(hwnd);

    // if we have the pointer, go to the message handler of the window

    // else, use DefWindowProc

    if (pWnd)
        return pWnd->WinMsgHandler(hwnd, uMsg, wParam, lParam);
    else
        return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

BOOL CBaseWindow::Create(DWORD dwStyles, RECT* rect)
{
    // Create the window


    // send the this pointer as the window creation parameter

    m_hwnd = CreateWindow(szClassName, szWindowTitle, dwStyles,
        rect->left, rect->top, rect->right - rect->left,
        rect->bottom - rect->top, NULL, NULL, hInstance,
        (void *)this);

    return (m_hwnd != NULL);
}
```

Conclusion

Although, the demo (and source) is a very simple and crude example of what you can do with a window wrapper class, I hope it will help make your Win32 programming projects easier to understand and your code more reusable.

If you can find more information on this subject, I would gladly add a links section to the article.

Credit where credit is due

- Class Members As Callbacks
- Creating a Win32 Window Wrapper Class

About the Author

Jason Henderson

I live in the middle of nowhere in Illinois, United States, and I am married and have three kids (all boys).

Occupation: Web Developer

Location: United States

Discussions and Feedback