Object oriented c++ win32?

Asked 14 years, 3 months ago Modified 4 years, 5 months ago Viewed 9k times

12

I want to create my own class to handle creating windows and the window procedure but I have noticed that the window procedure has to be static! I'm now wondering whether its possible to make the window procedure object oriented? I have read some tutorials on object oriented windows, but they always make the procedure static -.- whats the use in that? :/

Any links or info on how to get around this problem would be appreciated,

thanks

c++ winapi

Share Improve this question Follow

asked Aug 1, 2010 at 0:32

Kaije
2,631 6 38 42

See [Best method for storing this pointer for use in WndProc] (stackoverflow.com/questions/117792/...). – Matthew Flaschen Aug 1, 2010 at 0:40

2 It is for this reason that I've always wished that `WndProc` had a `void* user_data` param. It would make creating an object based wrapper just that much easier. – Evan Teran Aug 1, 2010 at 3:15

@Evan: yep, but it'd also have required someone *sane* to be in charge of designing the API... The Win32 API would've been a very different beast if that had been the case. – Stack Overflow is garbage Aug 1, 2010 at 13:42

2 And it would have required that the Windows API be written at a time AFTER C++ was created - the Windows APIs were created at around 1983 (and release in 1985), C++ was only created in 1983 and didn't achieve mainstream acceptance until the early 1990s (the ARM wasn't published 'til 1990). – Larry Osterman Aug 1, 2010 at 17:01

@Larry: in principle, sure. But the notion of tying some arbitrary user data to a call back for context is certainly not unique to c++. I've seen plenty of "ancient" APIs that have this functionality, which just so happens to jive very well with creating c++ wrappers. Oh well. – Evan Teran Aug 1, 2010 at 17:26

5 Answers

Sorted by: Highest score (default)

12

You can get around that by making the static WndProc delegate everything to the members:

```
// Forward declarations
class MyWindowClass;
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

std::map<HWND, MyWindowClass *> windowMap;

// Your class
```

```
class MyWindowClass  {
private:
  HWND m_handle;

  // The member WndProc
  LRESULT MyWndProc(UINT message, WPARAM wParam, LPARAM lParam) { /* ... */ }

public:
  MyWindowClass()
  {
    /* TODO: Create the window here and assign its handle to m_handle */
    /* Pass &WndProc as the pointer to the Window procedure */

    // Register the window
    windowMap[m_handle] = this;
  }
};

// The delegating WndProc
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  std::map<HWND, MyWindowClass *>::iterator it = windowMap.find(hWnd);
  if (it != windowMap.end())
    return it->second->MyWndProc(message, wParam, lParam);
  return 0;
}
```

Share Improve this answer Follow

answered Aug 1, 2010 at 0:42

Karel Petranek
15.1k 4 46 69

1 +1, but you might also want to check for WM_DESTROY there and remove the handle from the map. – SoapBox Aug 1, 2010 at 0:45

nice example, looks cool. I will try to implement it in my own, nice use of std::map to find the matching handle – Kaije Aug 1, 2010 at 0:49

@SoapBox It is incomplete in many ways, thanks for noting this one though. – Karel Petranek Aug 1, 2010 at 0:52

+1 for having a dictionary of HWND to MyWindowClass. I've seen methods before where people store pointers inside win32 structures, cast it, and dereference it directly. That's an extremely easy way to make your program vulnerable to hacks. – Merlyn Morgan-Graham Aug 1, 2010 at 2:00

It seems that `CreateWindowEx` will always return `NULL`, without giving a detailed error code from `GetLastError` (it returns `ERROR_SUCCESS`). In the delegating `WndProc` you should replace `return 0;` with `return DefWindowProc(hWnd, message, wParam, lParam);` and it'll fix this issue. – xian May 8, 2011 at 12:32

7

The general technique of allowing a window instance to be represented by as class instance is to make use of the SetWindowLongPtr and GetWindowLongPtr to associate your class instance pointer with the window handle. Below is some sample code to get you started. It may not compile without a few tweaks. It's only meant to be a reference.

Personally, I've stopped rolling my own window classes back a few years ago when I discovered ATL's CWindow and CWindowImpl template class. They take care of doing all this

mundane coding for you so can focus on just writing methods that handle window messages. See the example code I wrote up here.

Hope this helps.

```
class CYourWindowClass
{
private:
    HWND m_hwnd;

public:
    LRESULT WndProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
    {
        switch (uMsg)
        {
            case WM_CREATE: return OnCreate(wParam, lParam);
            case wM_PAINT: return OnPaint(wParam, lParam);
            case WM_DESTROY:
            {
                SetWindowLongPtr(m_hwnd, GWLP_USERDATA, NULL);
                m_hwnd = NULL;
                return 0;
            }
        }
        return DefWindowProc(m_hwnd, uMsg, wParam, lParam);

    }

    CYourWindowClass()
    {
        m_hwnd = NULL;
    }

    ~CYourWindowClass()
    {
        ASSERT(m_hwnd == NULL && "You forgot to destroy your window!");
        if (m_hwnd)
        {
            SetWindowLong(m_hwnd, GWLP_USERDATA, 0);
        }
    }

    bool Create(...) // add whatever parameters you want
    {
        HWND hwnd = CreateWindow("Your Window Class Name", "Your Window title",
dwStyle, x, y, width, height, NULL, hMenu, g_hInstance, (LPARAM)this);
        if (hwnd == NULL)
            return false;

        ASSERT(m_hwnd == hwnd);
        return true;
    }


    static LRESULT __stdcall StaticWndProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
    {
        CYourWindowClass* pWindow = (CYourWindowClass*)GetWindowLongPtr(hwnd,
GWLP_USERDATA);

        if (uMsg == WM_CREATE)
        {
            pWindow = ((CREATESTRUCT*)lParam)->lpCreateParams;
            SetWindowLongPtr(hwnd, GWLP_USERDATA, (void*)pWindow);
```

```
            m_hWnd = hwnd;
        }

        if (pWindow != NULL)
        {
            return pWindow->WndProc(uMsg, wParam, lParam);
        }

        return DefWindowProc(hwnd, uMsg, wParam, lParam);
    };


};
```

Share Improve this answer Follow

edited May 23, 2017 at 11:53
Community Bot
1 1

answered Aug 1, 2010 at 1:00
selbie
104k 15 107 182

Always wondered what the lpParam was for on the CreateWindowEx function, but i support this could be the way that makes it vulnerable, as somebody could use GetWindowLong to get your user data :P – Kaije Aug 1, 2010 at 2:21

3

Just to add to Brian's answer but for a win32 framework that's more beginner friendly take a look at Win32++. The library itself isn't as comprehensive in features compared to MFC or QT but that is a tradeoff the designer made at the beginning to keep the library easy to understand and simple to use.

If you're still interested in this topic, I highly encourage you to take a look at it since it uses yet another technique for saving the 'this' pointer by utilizing thread local storage.

Share Improve this answer Follow

answered Jan 15, 2011 at 4:38
greatwolf
20.8k 13 72 105

3

If you are looking for object oriented Win32 API then you should look to MFC and/or WTL.

Share Improve this answer Follow

edited May 6, 2020 at 10:44
Aykhan Hagverdili
29.7k 6 48 102

answered Aug 1, 2010 at 0:37
Brian R. Bondy
346k 125 601 640

Overkill solutions for many purposes. MFC may be *relatively* thin, but it's still a large API in its own right, and switching from one API to another is a lot of work. It's also completely unnecessary if all you want to do is use a few C++ classes of your own while coding for Win32. My own old "object framework" for Win32 was probably about 2 or 3 sides of code - little more than a base class, some initialisation and a main GetMessage/etc loop. – user180247 Aug 1, 2010 at 0:58

MFC is a nasty framework, but it's well supported and relatively well known in the Win32 world. – seand Aug 1, 2010 at 2:43

MFC may be overkill, but WTL is a joy. A very poorly documented joy... but a joy nonetheless. (well, relatively speaking with Win32 and all) – Dragontamer5788 Dec 5, 2014 at 21:03

1

You can use the window handle passed to the WindowProc to grab an object you've created for that particular window and delegate the event handling to that object.

e.g.

```
IMyWindowInterface* pWnd = getMyWindowObject(hWnd);
pWnd->ProcessMessage(uMsg, wParam, lParam);
```

Share Improve this answer Follow

answered Aug 1, 2010 at 0:37

Will A
24.9k 5 52 61

Sounds good, i noticed that the only unique thing in the procedure is the handle, but wasn't sure how to find my window through it. like in your example, the getMyWindowObject(hwnd), would that function consist of iterating through my open windows to see if the handle matches? because if so, if i was handling a WM_MOUSEMOVE or WM_TIMER, wouldn't that be tedious on the processor? – Kaije Aug 1, 2010 at 0:44

Your best bet it to use some form of hashtable to hash from HWND to pointers-to-your-window-object - that was the lookups are quick. Unless you've a pretty large number of windows open, I would expect a loop through all (HWND,object*) pairs would be quick enough tho'. – Will A Aug 1, 2010 at 0:53