

# A short guide to a better understanding of (the) windows

## 2. Subclass *-ing*

Radu Motisan  
Radu.motisan@gmail.com

### Introduction

Some time ago I had to deal with a modular project created in C#. It was quite big, and it needed low level coding for controlling the Wifi, the Phone and a few more on the Pocket PC.

Unfortunately it was a terrible piece of code. It was using a multiform-GUI, and tons of external DLLs for everything that the (incapable) C# / CF was not offering. Basically the project was what most of the C# projects are: A C# interface linked with external Win32 API Dll-s (written in C++).

So it had a Dll for the Phone interface (using RIL), a Dll for Wifi (using WZC), and it also had some references to that OpenNET, that IMHO I could call it as a "collection of everything that is usefull and handy goes to my library".

The problem was that at some point, the C# code was creating a "textbox" control. I needed something a bit customized to properly suit my needs. But by using the C# textbox, I could only add the dot after my object and see if I find something usefull in the popup list. Of course I couldn't find anything there... I needed support for a standard popup menu for that textbox, OR, at least, to have what the CAPEDIT class is offering. The textbox created with C# was a win32 window with the standard class "EDIT".

Just before the C# textbox was created, the C# code was loading an external general purpose DLL created in C++ win32 native API code. That was all I needed...

### #Define Subclassing?

A good definition would go as: "If class A inherits from class B, A is the subclass of B."

So for the "EDIT" control-class, I will try to keep the functionality that it already offers, and to add a few more features to it.

### How to do it?

Well first the idea is as described below:

Point A -> Dll is loaded by the C# code. This Dll calls the subclassing function

Point B -> the C# textbox is created

Just before the Point B, we need to have the subclassing already done. To simplify things for this article, lets convert everything to a C++ program (no need for external Dlls for this article):

Point A becomes the Subclassing function itself.

Point B becomes: `HWND hEnhancedEdit = CreateWindow(TEXT("edit"),...`

So the C++ code will be something close to:

```
//Do the Subclass
SubClassControl (TEXT("EDIT"), hInstance);
//Create our editbox
HWND editBox = CreateWindow(TEXT("EDIT"), TEXT("test"),
    WS_VISIBLE | WS_CHILD | WS_BORDER ,
    20,20,60,20,
    g_mainWnd, (HMENU)1041,hInstance,0);
```

Where the SubClassControl is:

```
BOOL SubClassControl(TCHAR *szClassName, HINSTANCE hInst)
{
    WNDCLASS wndclass = {0};

    // get "old" class info
    if (GetClassInfo(hInst,szClassName, &wndclass) == 0) return
FALSE;
    oldControlProc = wndclass.lpfnWndProc;

    // get rid of the "old" class
    UnregisterClass(szClassName, hInst);

    // handle all events in our proc
    wndclass.lpfnWndProc = (WNDPROC) SubClassControlProc;
    return RegisterClass(&wndclass);
}
```

The beauty of this comes where we can add our custom callback for the control SubClassControlProc:

```
WNDPROC oldControlProc;

LRESULT CALLBACK SubClassControlProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
    switch (message)
    {
```

```
        case WM_LBUTTONDOWN:
            MessageBeep(0);
            break;
        default:
            return CallWindowProc(oldControlProc, hWnd, message,
wParam, lParam);
    }
    return 0;
}
```

In this form, when the user taps on the editbox, it will hear the sound created by MessageBeep. Easily this can be modified to offer the popup support for this messagebox.

This applies to other native win32 controls too. It's a powerful technique that can help you add a callback to an already defined object for interception reasons, it can help you enrich the functionality of a control, and many other...

There is an even more advanced way of "subclassing" in which you can actually "hack" into other processes' windows, to add your own code.

Some examples:

- The Teksoft Smartbar: the user can use the keyboard to move the selection from the Smartbar to the Today and FROM THE TODAY BACK TO THE SMARTBAR, by subclassing the today screen itself.
- The O2 Active User Interface v4.0: the user can hear sounds while pressing keys over the Today Screen.

As always, feedback is welcomed.

Cheers,

Radu Motisan

Saturday, December 30, 2006