Private classes, superclassing, and global subclassing

devblogs.microsoft.com/oldnewthing/20100215-00

February 15, 2010

Raymond Chen

In the suggestion box, A. Skrobov asks why it's impossible to superclass `WC_DIALOG`, but the example that follows is not actually superclassing.

> When I register my own class under this atom, and leave NULL in WNDCLASS.hInstance, Windows fills it in for me. Then I have two distinct classes registered: (0,WC_DIALOG) and (hMyInstance,WC_DIALOG), and DialogBox functions all use the first one.

This question is a bit confused, since it says that the goal is to superclass the dialog class, but registering `WC_DIALOG` is not superclassing.

First, I'll refer everyone to this MSDN article which describes the various ways of manipulating a window class: Subclassing, superclassing, and global subclassing.

To superclass the dialog class, you retrieve information about the class by calling `GetClassInfo` and then register a *new* class based on the original class. But you don't need to go to all that effort to superclass the dialog class, because you already know what you need to know: The number of extra bytes is `DLGWINDOWEXTRA`, and the dialog procedure is `DefDlgProc`. You can just register your superclass directly, as we saw last time.

Superclassing is done by registering your custom class under a different name, and using that class name if you want to obtain the new behavior. On the other hand, the question about talks about registering a class under the same name as the original (namely, `WC_DIALOG`). This isn't subclassing, nor is it superclassing, nor is it even global subclassing.

Before continuing the discussion, I'll first address the issue of leaving `NULL` in `WNDCLASS.hInstance`: The value `NULL` for the instance handle is not legal when registering a class. Each class is associated with a module instance, and `NULL` is not a module instance. The window manager autocorrects this mistake by registering the class under the module corresponding to the executable. This is the same special-case behavior you get if you call `GetModuleHandle(NULL)`, so it's not something completely out of the blue. It looks like A. Skrobov is being confused by the window manager's attempt to *do what you mean*. So much for being helpful.

Okay, back to the original problem. Recall that the `HINSTANCE` member of the `WNDCLASS` structure is used to specify the class namespace. If you register a class against the handle of the current executable, then in order to create a window with that class, you need to create it with that same instance handle.

Now we can put all the pieces together: Registering the class with `WNDCLASS.hInstance = NULL` is autocorrected to registering it with `WNDCLASS.hInstance = GetModuleHandle(NULL)`, which places the class in the window class namespace of the current module. This is a separate class from the system dialog class, which is registered against `GetModuleHandle(TEXT("USER32"))`. The two are registered against different modules, so they live independent lives. They just happen to have the same name.

As we learned a few years ago, the instance handle you pass to the `CreateWindow` (or related) function is used to look up the window class, and as we also learned, the `HINSTANCE` you pass to the `DialogBox` (or related) function is used to look up the template as well as to create the frame window. The class name comes from the template, and if you didn't specify an explicit class in your template, then the dialog manager will use `WC_DIALOG`.

You now have all the pieces necessary to understand what is going on. When you register the class against your executable's instance, you need to use that same instance when creating the dialog box so that your private class is found instead of the global one.

To show how this all fits together, I've written a little program which registers a private class which happens to have the name `WC_DIALOG` and then uses it to create a dialog box.

```
// scratch.rc
#include <windows.h>
// A pointless dialog box, for illustration only
1 DIALOG 0,0,150,50
STYLE DS_MODALFRAME | DS_SHELLFONT | WS_POPUP | WS_VISIBLE |
    WS_CAPTION | WS_SYSMENU
CAPTION "Pointless"
FONT 8, "MS Shell Dlg"
BEGIN
    DEFPUSHBUTTON "Cancel",IDCANCEL,50,18,50,14
END
// scratch.cpp
#include <windows.h>
LRESULT CALLBACK
SuperDlgProc(HWND hwnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uiMsg) {
  case WM_ERASEBKGND:
    return DefWindowProc(hwnd, uiMsg, wParam, lParam);
  }
  return DefDlgProc(hwnd, uiMsg, wParam, lParam);
}
INT_PTR CALLBACK
DlgProc(HWND hwnd, UINT wm, WPARAM wParam, LPARAM lParam)
{
  switch (wm) {
  case WM_INITDIALOG: return TRUE;
  case WM_CLOSE: EndDialog(hwnd, 0); return TRUE;
  }
  return FALSE;
}
int CALLBACK
WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
        LPSTR pszCmdLine, int nShowCmd)
{
  WNDCLASS wc;
  wc.style = 0;
  wc.lpfnWndProc = SuperDlgProc;
  wc.cbClsExtra = 0;
  wc.cbWndExtra = DLGWINDOWEXTRA;
  wc.hInstance = hinst;
  wc.hIcon = NULL;
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = (HBRUSH)(COLOR_INFOBK + 1);
  wc.lpszMenuName = NULL;
  wc.lpszClassName = WC_DIALOG;
  if (RegisterClass(&wc))
    DialogBox(hinst, MAKEINTRESOURCE(1), NULL, DlgProc);
  return 0;
}
```

The dialog template is itself entirely unremarkable; it looks like any old dialog template.

Our superclass takes the regular dialog box class and gives it a custom background color, namely `COLOR_INFOBK`.

The program registers this private version of `WC_DIALOG` and creates a dialog box based on it. Since we passed the same `HINSTANCE` in the `WNDCLASS.hInstance` as we did to `DialogBox`, the lookup of the `WC_DIALOG` class will find our private version and use it instead of the global version.

Raymond Chen

Follow