When you subclass a window, it's the original window procedure of the window you subclass you have to call when you want to call the original window procedure

devblogs.microsoft.com/oldnewthing/20090507-00

May 7, 2009

Raymond Chen

When you subclass a window, you set the window procedure to a function of your choosing, and you remember the original window procedure so you can pass it to the `CallWindowProc` function when your subclass function wants to pass the message to the original window procedure. For example, if you subclass a window like this:

```
SubclassWidgetDialog(HWND hdlgWidget)
{
  HWND hwndButton = GetDlgItem(hdlgWidget, …);
  wndProcOrig = (WNDPROC)
     SetWindowLongPtr(hwndButton, GWLP_WNDPROC, (LONG_PTR)SubclassWndProc);
  // -or-
  // wndprocOrig = SubclassWindow(hwndButton, SubclassWndProc);
  …
}
```

then your subclass function should go something like this:

```
LRESULT CALLBACK SubclassWndProc(
    HWND hwnd, UINT wm, WPARAM wParam, LPARAM lParam)
{
  switch (wm) {
  …
  default:
    return CallWindowProc(wndprocOrig, hwnd, wm, wParam, lParam);
  }
}
```

The window procedure you pass to `CallWindowProc` is the one which was the window procedure of that window before you subclassed it, not the window procedure from some other window. In the same way that when you create a derived C++ class, you pass method calls along to your base class, not to somebody else's base class:

```
class DerivedClass : public BaseClass {
 …
 // override base class: do some extra stuff
 int Method(int param)
 {
  ExtraDerivedStuff();
  return BaseClass::Method(param);
 }
};
```

When you are finished with your customization in `DerivedClass::Method`, you let the base class do what normally would have happened if you hadn't overridden the method in the first place by calling `BaseClass::Method` and not by calling `SomeOtherClass:Method`.

This sounds blatantly obvious, but you'd be surprised how often people mess this up. For example, if you subclass multiple widget dialogs, you have to save the old window procedure in a different place for each one, because each button may have had a different window procedure by the time you got around to subclassing it. For example, one of them might be a plain button, whereas another of them was subclassed in order to provide a tooltip. If you make `wndprocOrig` a global or static variable, then you're assuming that every widget button has the same window procedure. You are subclassing a window and forgetting to handle the case where the window is already subclassed! You forgot that somebody else could have done exactly what you're doing.

There is a popular commercial program that comes preinstalled on many computers which creates a common file open dialog box and subclasses both the file name combo box and the file type combo box, and figures that, well, since they're both combo boxes, they must have the same window procedure, right? Unfortunately, there's no guarantee that they do, because the common file dialog is free to subclass them in order to provide custom behavior like autocomplete. It so happens that the program grabs the window procedure from the subclassed combo box window and uses it for all combo boxes. (These are probably the same folks who would have called the `BOZOSLIVEHERE` function if given the chance.)

This makes for very exciting crashes when they take the original window procedure from the subclassed combo box and use it for the other, unsubclassed, combo box. The subclass window procedure finds itself handed a window *that it never subclassed*. As a result, it not only can't perform its own subclass behavior, but can't even just fall back and say "Well, I can't do my custom stuff, so I'll just forward to the original window procedure" since it can't figure out what the original window procedure was either. It's the window manager version of writing this strange C++ code:

```
class SiblingClass : public BaseClass { … };
class DerivedClass : public BaseClass {
 …
 // override base class: do some extra stuff
 // then pass method to the WRONG base class
 int Method(int param)
 {
  ExtraDerivedStuff();
  return ((SiblingClass*)this)->Method(param);
 }
};
```

When you subclass a window, and you want to call the original window procedure, make sure you call the correct original window procedure: The one that was the window procedure of that window before you subclassed it.

Raymond Chen

Follow