When will the window manager destroy a menu automatically, and when do I need to do it manually?

devblogs.microsoft.com/oldnewthing/20100527-00

May 27, 2010

Raymond Chen

Our old friend Norman Diamond wonders when you are supposed to destroy a menu and when you are supposed to let Windows destroy it.

The rules for when the window manager implicitly destroys menus are actually not that complicated.

- If a window is destroyed, the menus attached to the window are also destroyed:
  - Attached as the menu bar (`GetMenu` / `SetMenu`)
  - Attached as the system menu (`GetSystemMenu`)
- If a menu is destroyed, its submenus are also destroyed.
- If you replace a `MIIM_SUBMENU` submenu, the old menu is destroyed.
- If you pass `bRevert = TRUE` to `GetSystemMenu` , then the old system menu is destroyed and a clean system menu is created in its place.

Outside of the above situations, you are on your own.

Of course, when I write that "you are on your own" I do not mean that "every code which sees a menu is responsible for destroying it." If that were the case, you would have a disaster as the slightest passing breeze would cause people to call `DestroyMenu` all over the place. Rather, I mean that in all other cases, you need to "work it out amongst yourselves" who is responsible for destroying the menu. Typically, the person who creates the menu takes responsibility for destroying it, although that responsibility can be handed off based on mutual agreement between the creator and another component.

The original question did include a misunderstanding:

> If the old object belonged to a window class, and we destroy the old object, how do we know that other windows of the same class aren't going to get in trouble?

The mistaken belief here is that each window of a class shares the same menu. If that were true, then if a program created two windows of the same class, modifications to one window's menu would affect the other. You can see that this is not true by inspection, or at least it was

easier back in 1995. On Windows 95, open two Explorer windows, and set them into different views. The two windows now have different menus: One of them has a bullet next to the *Large Icons* menu item, whereas the other has a bullet next to *Details*.

When you register a window class, you pass in the menu you want, but only in the form of a template:

```
WNDCLASS wc;
...
wc.lpszMenuName = MAKEINTRESOURCE(...);
```

There is no menu yet, just a description of how to create a menu when the time comes. When you create a window from this class, the window manager initializes the menu by doing the equivalent of

```
SetMenu(hwnd, LoadMenu(pWndClass->hInstance,
                       pWndClass->lpszMenuName));
```

Each window gets a fresh menu from the specified menu template. Once that's done, you can change it all you want; it won't affect any the menus associated with any other windows.

The system menu works the same way: Every window starts out with a default system menu, and when you call `GetSystemMenu` with `bRevert = FALSE`, you get a handle to that system menu, which you can modify to your heart's content without affecting any other menus. System menus have this additional wrinkle where you can pass with `bRevert = TRUE` to ask the window manager to destroy the current system menu and replace it with a fresh new default system menu.

Exercise: How would you accomplish the logical equivalent of `GetSystemMenu(TRUE)` for the menu bar menu?

Bonus chatter: While the system menu certainly behaves as I described it above, there's actually a little bit of optimization going on under the hood. We'll look at that next time.

Raymond Chen

Follow