The forgotten common controls: The ShowHideMenuCtl function

devblogs.microsoft.com/oldnewthing/20070710-00

July 10, 2007

Raymond Chen

The `ShowHideMenuCtl` function is one of those functions everybody tries to pretend doesn't exist. You thought `MenuHelp` was bad; `ShowHideMenuCtl` is even worse.

The idea behind `ShowHideMenuCtl` was that you had a window with a menu as well as controls, and some of the menu items were checkable, indicating whether the corresponding control should be shown. For example, on your View menu you might have options named Toolbar or Status Bar. If the user checks Toolbar, then the toolbar is shown in the main window; if the user unchecks Toolbar, then the toolbar is hidden.

The parameters to the `ShowHideMenuCtl` function are a window (the window on which you want to operate), a menu identifier (the menu item you wish to toggle), and a mysterious array of integers. Everything hangs on that mysterious array of integers, which takes the following form (expressed in pseudo-C):

```
struct MENUCONTROLINTS {
 int idMenu;
 int idControl;
};
struct SHOWHIDEMENUCONTROLINTS {
 int idMainMenu;
 HMENU hmenuMain;
 MENUCONTROLINTS rgwMenuControl[];
};
```

The `MENUCONTROLINTS` structure is easier to describe. It merely establishes the correspondence between a menu item and the control that will be shown or hidden. (Exercise: Why do we need two integers? Why can't we just give the menu item and the control the same ID?) The array of `MENUCONTROLINTS` structures is terminated by a pair whose `idMenu` is zero.

The tricky bit is the first two entries, `idMainMenu` and `hmenuMain`. The `hmenuMain` is the handle to the main menu for the window, and the `idMainMenu` is the item on the menu corresponding to the "Hide menu" entry on the main menu. (That's why `hmenuMain` need to be passed explicitly. We would normally use `GetMenu(hwnd)` to get the handle to the main menu, but if we've removed it, then `GetMenu(hwnd)` will return `NULL`.) If you don't want to have a "Hide menu" option, you can just put a dummy value in the `idMainMenu` slot that doesn't correspond to any menu item. (The value `-1` is probably most convenient for this. Don't use zero since it terminates the list!)

When you call the `ShowHideMenuCtl` function, it searches for the menu item you specified and toggles the check mark next to that item. What happens next depends on what type of item was found.

- If the item is `idMainMenu`, then the main menu is attached to or removed from the window (by using the `SetMenu` function, of course), corresponding to the check box.
- If the item is `idMenu`, then the corresponding control is shown or hidden (by using the `ShowWindow` function, of course), corresponding to the check box.

That's all there is to it. The rest is up to you. For example, when a control is shown or hidden, it's still up to your program to relayout the visible controls to account for the new window visibility state. For example, if the user shows the toolbar, then the other controls need to move out of the way to make room for the toolbar. The `ShowHideMenuCtl` function can't do this for you since it has no idea what your window layout is.

Let's put this information into practice. Start with our scratch program and make the following changes;

```
HMENU g_hmenuMain;
INT rgiMenu[] = {
    100, 0,
    101, 200,
    0, 0,
};
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
    /* We'll talk about this line more later */
    rgiMenu[1] = (INT)GetMenu(hwnd);
    CreateWindow(TEXT("Button"), TEXT("Sample"),
                 WS_CHILD | BS_PUSHBUTTON, 0, 0, 100, 100,
                 hwnd, IntToPtr_(HMENU, 200), g_hinst, 0);
    return TRUE;
}
void
OnDestroy(HWND hwnd)
{
    if (!GetMenu(hwnd))
        DestroyMenu(IntToPtr_(HMENU, rgiMenu[1]));
    PostQuitMessage(0);
}
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
    switch (id) {
    case 100:
    case 101: ShowHideMenuCtl(hwnd, id, rgiMenu); break;
    }
}
HANDLE_MSG(hwnd, WM_COMMAND, OnCommand);
BOOL
InitApp(void)
{
    ....
    wc.lpszMenuName = MAKEINTRESOURCE(1);
    ....
}
/* add to resource file */
1 MENU PRELOAD
BEGIN
    POPUP "&View"
    BEGIN
        MENUITEM "&Menu Bar", 100, CHECKED
        MENUITEM "&Button", 101
    END
END
```

Most of the changes are just setting up. We attach a menu to our window with two options, one to hide and show the menu bar, and one to hide and show our custom button. Since our window starts out with the menu bar visible and the button hidden, our menu template checks the "Menu Bar" item but not the "Button" one.

The `OnCreate` function finishes setting up up the `rgiMenu` array by putting the main menu's handle into index 1 in the array of integers, which corresponds to `hmenuMain` in our pseudo-structure. The `OnDestroy` function destroys the menu if it isn't attached to the window, since menus attached to a window are destroyed automatically when the window is destroyed. The magic happens in the `OnCommand` handler. If the user picked one of our two menu items, then we ask `ShowHideMenuCtl` to hide and show the button or menu.

The tricky bit is setting up our `rgiMenu`. Let's break down those integers.

100	Menu identifier for hiding and showing the menu bar
0	Placeholder (receives main menu handle in `OnCreate` handler)
101	Menu identifier for hiding and showing the menu bar
200	Control ID for the button that is shown and hidden (passed to the `CreateWindow` function)
0, 0	List terminator

When you run this program, you can use the "Button" menu option to hide and show the button, and you can use the "Menu Bar" menu option to hide and show the window's main menu. Erm, no wait, you can't use it to show the main menu, because the main menu is hidden! Naturally, if your program uses the ability to hide the main menu, you need to provide some alternate mechanism for bringing the main menu back, say via a hotkey or by adding an option to the System menu.

Okay, now back to that line in the `OnCreate` function that I promised to talk about. If you have been paying attention, alarm bells should have gone off in your head at the line `rgiMenu[1] = (INT)GetMenu(hwnd);` because we are casting an `HMENU` to an integer. On 64-bit machines, a `HMENU` is a 64-bit value, but integers are only 32-bit. This cast truncates the handle value and consequently is not 64-bit safe. Since the `ShowHideMenuCtl` function requires an array of integers, you're stuck. You can't shove a 64-bit menu handle into a 32-bit integer. The `ShowHideMenuCtl` function is fundamentally flawed; it is not 64-bit compatible.

Fortunately, nobody uses the `ShowHideMenuCtl` function anyway. Its functionality is so simple, most programs have already written a function that does roughly the same thing, and since you have to write the layout code anyway, the `ShowHideMenuCtl` function doesn't really save you very much effort anyway. Like `MenuHelp`, the function is entirely vestigial and isn't something you should be tempted to use in any modern program. It's a leftover from the days of 16-bit Windows.

Why does such a confusing function exist at all? Well, the shell team thought they were doing you a favor by providing this function back in the 16-bit days. This was originally an internal function used by (I think it was) File Manager, but since it solved a more general problem, the function was exported and documented. In the intervening years, the problem it addressed has been solved in other ways, and the introduction of 64-bit Windows rendered the original solution unworkable anyway, but the function and the code behind it must still linger in the system for backwards compatibility purposes.

The shell team learned its lesson. It no longer exports every little helper function and custom control for third parties to use. If a future version of Windows no longer needs the helper function, or if a redesign of Windows Explorer removes the need for that custom control (or worse, changes the behavior of that custom control), the shell would still have to carry all the code around for the unused function or control because a function, once documented, becomes a continuing support burden.

Raymond Chen

Follow