How do I make it so that users can copy static text on a dialog box to the clipboard easily?

devblogs.microsoft.com/oldnewthing/20120301-00

March 1, 2012

Raymond Chen

Given that you have a Win32 dialog box with static text in an `LTEXT` control, how do you make it so that users can easily copy that text to the clipboard?

The traditional solution is to create a borderless read-only edit control (which draws as static text by default). Add it to the tab order by setting the `WS_TABSTOP` style, and maybe even give it a keyboard accelerator for accessibility.

Starting in Windows Vista, version 6 of the common controls provides an alternative. (A less accessible alternative, mind you.) Static text controls automatically copy their contents to the clipboard when you double-click them if you set the `SS_NOTIFY` style.

Let's try it:

```
#include <windows.h>
#include <windowsx.h>
#include <commctrl.h>
#pragma comment(linker, \
 "\"/manifestdependency:type='Win32' "\
 "name='Microsoft.Windows.Common-Controls' "\
 "version='6.0.0.0' "\
 "processorArchitecture='*' "\
 "publicKeyToken='6595b64144ccf1df' "\
 "language='*'\"")
INT_PTR CALLBACK DlgProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 switch (uMsg) {
 case WM_INITDIALOG:
  return TRUE;
 case WM_COMMAND:
  switch (GET_WM_COMMAND_ID(wParam, lParam)) {
  case IDCANCEL:
   EndDialog(hdlg, 0);
   break;
  }
 }
 return FALSE;
}
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
 return DialogBox(hinst, MAKEINTRESOURCE(1), NULL, DlgProc);
}
// resource file
#include <windows.h>
1 DIALOG 50, 50, 100, 50
STYLE DS_SHELLFONT | WS_SYSMENU
CAPTION "Sample"
FONT 8, "MS Shell Dlg"
BEGIN
 LTEXT "Sample text 1",100,10,10,80,10,SS_NOTIFY
 LTEXT "Sample text 2",101,10,20,80,10,SS_NOTIFY
 LTEXT "Sample text 3",102,10,30,80,10
END
```

Run this program and double-click on the text controls, and observe that the text gets copied to the clipboard, or at least it does for the first two, since I set the `SS_NOTIFY` style on them.

Now, when the *double-click to copy* feature was added to the static control, there was no way to suppress it. The `STN_DBLCLK` notification is documented as ignoring its return code, so it would be a compatibility problem if suddenly it started studying its return code so that the

parent could respond “No, I handled the click, don’t do your default action.” Instead, if you want to disable the *double-click to copy* feature on a `SS_NOTIFY` static control, you have to subclass the static control and eat the clicks yourself.

```
LRESULT CALLBACK SuppressCopyOnClick(
    HWND hwnd, UINT uMsg, WPARAM wParam,
    LPARAM lParam, UINT_PTR uIdSubclass, DWORD_PTR dwRefData)
{
 switch (uMsg) {
 case WM_LBUTTONDBLCLK: return 0; // eat the double-click
 case WM_NCDESTROY:
  RemoveWindowSubclass(hwnd, SuppressCopyOnClick, uIdSubclass);
  break;
 }
 return DefSubclassProc(hwnd, uMsg, wParam, lParam);
}
INT_PTR CALLBACK DlgProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 switch (uMsg) {
 case WM_INITDIALOG:
  SetWindowSubclass(GetDlgItem(hdlg, 101),
                    SuppressCopyOnClick, 0, 0);
  return TRUE;
 ...
```

“Why would you add a style that enabled a feature, and then disable the feature?”

Maybe you want some other aspects of the feature but not the copy-on-double-click behavior. Maybe somebody else is adding the `SS_NOTIFY` style behind your back. For example, a UI framework might add it automatically to all static controls.

And actually, in that UI framework case, you probably want the `STN_DBLCLK` notification to be fired when a double-click occurs, because you added an `OnDoubleClick` handler to your class. You just don’t want the copy-to-clipboard behavior. We can fix that by firing the notification in our subclass procedure.

```
 case WM_LBUTTONDBLCLK:
  if (GetWindowStyle(hwnd) & SS_NOTIFY) {
   FORWARD_WM_COMMAND(GetParent(hwnd), GetDlgCtrlID(hwnd), hwnd,
                      STN_DBLCLK, SendMessage);
  }
  return 0; // message handled
```

To illustrate this change, we’ll make our dialog box beep when it gets a double-click notification. In real life, of course, you would do whatever you want to happen on the “double click on a static control” event. Actually, in real life, the code that responds to the `STN_DBLCLK` lives inside your framework, and it turns around and raises an `OnDoubleClick` event, but for simplicity, we’ll just code it inline.

```
case WM_COMMAND:
    switch (GET_WM_COMMAND_ID(wParam, lParam)) {
    case IDCANCEL:
       EndDialog(hdlg, 0);
       break;
    case 100:
    case 101:
    case 102:
     switch (GET_WM_COMMAND_CMD(wParam, lParam)) {
     // Obviously we would do something more interesting here
     case STN_DBLCLK: MessageBeep(MB_OK); break;
     }
    }
```

Each of the static controls on the dialog behaves differently. The first one is `SS_NOTIFY` with no subclassing, so double-clicking copies the text to the clipboard and also beeps. The second one is `SS_NOTIFY` with subclassing to disable the copy-to-clipboard, so double-clicking merely beeps. And the third one doesn't have the `SS_NOTIFY` style at all, so it neither copies the next nor responds to double-click.

Raymond Chen

Follow