Remember what happens when you broadcast a message

devblogs.microsoft.com/oldnewthing/20060612-00

June 12, 2006

Raymond Chen

Occasionally I catch people doing things like broadcasting a `WM_COMMAND` message to all top-level windows. This is one of those things that is so obviously wrong I don't see how people even thought to try it in the first place.

Suppose you broadcast the message

```
SendMessage(HWND_BROADCAST, WM_COMMAND, 100, 0);
```

What happens?

Every top-level window receives the message with the same parameters, and every top-level window starts interpreting those parameters in their own idiosyncratic way. As you know (since you've written them yourself), each window procedure defines its own menu items and child windows and there is no guarantee that command 100 will mean the same thing to each window. A dialog box with the template

```
#define IDC_USEDEFAULT 100
...
    AUTORADIOBUTTON "Use &default color",
                    IDC_USEDEFAULT, 14, 38, 68, 10, WS_TABSTOP
```

would interpret the message as

id	=	IDC_USEDEFAULT (100)
command	=	BN_CLICKED (0)
window	=	NULL (0) — illegal parameter

Depending on how the dialog procedure is written, it might try to send a message back to the button control (and fail since you passed `NULL` as the window handle), or it might update some dialog state like disabling the color customization controls (since it was told that the user clicked the "User default color" radio button).

Another dialog box might have the template

```
#define IDC_CHANGE 100
...
    PUSHBUTTON      "C&hange", IDC_CHANGE, 88, 95, 50, 14
```

This dialog procedure would interpret the message as

id	=	IDC_CHANGE (100)
command	=	BN_CLICKED (0)
window	=	NULL (0) — illegal parameter

The reaction would probably be to apply the changes that were pending in the dialog.

Meanwhile, another window might have a menu that goes like this:

```
#define IDC_REFRESH 100
...
        MENUITEM "&Refresh", IDC_REFRESH
```

It is going to interpret the message as the user having selected "Refresh" from the window menu.

id	=	IDC_REFRESH (100)
command	=	0 — illegal parameter, must be 1 for menu items
window	=	NULL (0)

Not only is the command code invalid for a menu item, the window might be in a state where the program had disabled the "Refresh" option. Yet you sent the window a message as if to say that the user selected it anyway, which is impossible. Congratulations, you just presented the program with an impossible situation and it very well may crash as a result. For example, the program may have disabled the "Refresh" option since there is no current object to refresh. When you send it the "Refresh" command, it will try to refresh the current object and crash with a null pointer error.

Obviously, then, you cannot broadcast the `WM_COMMAND` message since there is no universal meaning for any of the command IDs. A command ID that means "Refresh" to one window might mean "Change" to another.

The same logic applies to nearly all of the standard Windows messages. The ones that are actually designed to be broadcast are as follows:

WM_SYSCOLORCHANGE
WM_SETTINGCHANGE (= WM_WININICHANGE)
WM_DEVMODECHANGE
WM_FONTCHANGE
WM_TIMECHANGE
WM_DDE_INITIATE

If you try to broadcast a message in the `WM_USER` or `WM_APP` ranges, then you're even crazier than I thought. As we've already seen, the meaning of window messages in those ranges are defined by the window class or the application that created the window. Not only are the parameters to the message context-sensitive, the message itself is! This means that sending a random window a `WM_USER+1` message (say) will result in extremely random behavior. (We saw this before in the context of broadcasts, but it applies to directed delivery, too.) If it's a dialog box, it will think you sent a `DM_SETDEFID` message, and you just changed that dialog's default ID. If it's a common dialog box, it will think you sent a `WM_CHOOSEFONT_GETLOGFONT` message, and if you're lucky, it will crash trying to return the `LOGFONT` through an invalid pointer. (If you're not lucky, the parameter you passed will happen to be a valid pointer and the program will merely corrupt its own memory in some strange way, only to behave erratically later on.) If it's a tooltip control, then you just sent it the `TTM_ACTIVATE` message and you just manipulated the tooltip's activation state.

The same caution applies, using the same logic, to sending messages without universal meaning to windows whose window class you do not have an interface contract with. For example, I'll see people sending the `PSM_PRESSBUTTON` message to a window on the blind-faith assumption that it is a property sheet.

Remember, then, that when you send a message to a window, you need to be sure that the window will interpret it in the manner you intend.

Raymond Chen

Follow