Restating the obvious about the WM_COMMAND message

devblogs.microsoft.com/oldnewthing/20060302-10

March 2, 2006

Raymond Chen

I'm satisfied with the MSDN documentation for the `WM_COMMAND` message, but for the sake of mind-numbing completeness, I'm going to state the obvious in the hope that you, dear readers, can use this technique to fill in the obvious in other parts of MSDN.

The one-line summary of the `WM_COMMAND` message says, "The WM_COMMAND message is sent when the user selects a command item from a menu, when a control sends a notification message to its parent window, or when an accelerator keystroke is translated." In a nutshell, there are three scenarios that generate a `WM_COMMAND` message, namely the three listed above. You want to think of the menu and accelerator scenarios of the `WM_COMMAND` message as special cases of the control scenario.

The high-order word of the `wParam` parameter "specifies the notification code if the message is from a control". What does "control" mean here? Remember that you have to take things in context. The `WM_COMMAND` message is being presented in the context of Win32 in general, and in the context of the window manager in particular. Windows such as edit boxes, push buttons, and list boxes are commonly called "controls", as are all the window classes in the "common controls library". In the world of the window manager, a "control" is a window whose purpose is to provide some degree of interactivity (which, in the case of the static control, might be no interactivity at all) in the service of its parent window. The fact that the `WM_COMMAND` is used primarily in the context of dialog boxes further emphasizes the point that the term "control" here is just a synonym for "child window".

What does "notification code" mean here? Control notification codes are arbitrary 16-bit values defined by the control itself. By convention, they are named `xxN_xxxx`, where the "N" stands for "notification". Be careful, however, not to confuse this with notification codes associated with the `WM_NOTIFY` message. Fortunately, every notification code specifies in its documentation whether it arrives as a `WM_COMMAND` notification or a `WM_NOTIFY` notification. A modern control designer is more likely to use `WM_NOTIFY` notifications since they allow additional information to be passed with the notification. The `WM_COMMAND` message, by comparison, passes only the notification itself; the other parameters to the `WM_COMMAND` message are forced, as we'll see below. If `WM_NOTIFY` is superior to

`WM_COMMAND` , why do some controls use `WM_COMMAND` ? Because `WM_NOTIFY` wasn't available until Windows 95. Controls that were written prior to Windows 95 had to content themselves with the `WM_COMMAND` message.

"If the message is from an accelerator, this value [the high-order word of the `wParam` parameter] is 1." Remember, we're still in the context of the window manager, and particular in the context of the `WM_COMMAND` message. The accelerator here refers to messages generated by the call to `TranslateAccelerator` in the message loop.

"If the message is from a menu, this value is zero." If the `WM_COMMAND` mesage was triggered by the user selecting an item from a menu, then the high-order word of the `wParam` is zero.

The low-order word of the `wParam` parameter "specifies the identifier of the menu item, control, or accelerator." The identifier of a menu item or accelerator is the command code you associated with it in your menu or accelerator template or (in the case of a menu item) when you manually created the menu item with a function like `InsertMenuItem` . (You probably named your menu item identifiers and accelerator identifiers `IDM_something` .) The identifier of a control is determined by the creator of the control; recall that the `hMenu` parameter to the `CreateWindow` and `CreateWindowEx` functions is treated as a child window identifier if you're creating a child window. It is that identifier that the control identifier. (You can retrieve the identifier for a control by calling the `GetDlgCtrlID` function.)

Finally, the `lParam` parameter is the "handle to the control sending the message if the message is from a control. Otherwise, this parameter is NULL." If the notification is generated by a child window (with a notification code appropriate for that child window, obviously), then that child window handle is passed as the `lParam` . If the notification is generated by an accelerator or a menu, then the `lParam` is zero.

Notice that nearly all of the parameters to the `WM_COMMAND` message are forced, once you've decided what notification you're generating.

If you are generating a notification from a control, you must pass the notification code in the high word of the `wParam` , the control identifier in the low word of the `wParam` , and the control handle as the `lParam` . In other words, once you've decided that the `hwndC` window wants to send a `CN_READY` notification, you have no choice but to type

```
SendMessage(GetParent(hwndC), WM_COMMAND,
            MAKEWPARAM(GetDlgCtrlID(hwndC), CN_READY),
            (LPARAM)hwndC);
```

In other words, all control notifications take the form

```
SendMessage(GetParent(hwndC), WM_COMMAND,
            MAKEWPARAM(GetDlgCtrlID(hwndC), notificationCode),
            (LPARAM)hwndC);
```

where `hwndC` is the control generating the notification and `notificationCode` is the notification code. Of course, you can use `PostMessage` instead of `SendMessage` if you would rather post the notification rather than sending it.

The other two cases (accelerators and menus) are not cases you would normally code up, since you typically let the `TranslateAccelerator` function deal with accelerators and let the menu system deal with menu identifiers. But if for some reason, you wanted to pretend that the user had typed an accelerator or selected a menu item, you can generate the notification manually by following the rules set out in the documentation.

```
// simulate the accelerator IDM_WHATEVER
SendMessage(hwnd, WM_COMMAND,
            MAKEWPARAM(IDM_WHATEVER, 1),
            0);
```

Here, `hwnd` is the window that you want to pretend was the window passed to the `TranslateAccelerator` function, and `IDM_WHATEVER` is the accelerator identifier.

Simulating a menu selection is exactly the same, except that (according to the rules above), you set the high-order word of the `wParam` to zero.

```
// simulate the menu item IDM_WHATEVER
SendMessage(hwnd, WM_COMMAND,
            MAKEWPARAM(IDM_WHATEVER, 0),
            0);
```

Here, `hwnd` is the window associated with the menu. A window can be associated with a menu either by being created with the menu (having passed the menu handle to the `CreateWindow` or `CreateWindowEx` function explicitly, or having it done implicitly by including it with the class registration) or by having been passed explicitly as the window parameter to a function like `TrackPopupWindow` .

One significant difference between the accelerator/menu case and the control notification case is that accelerator and menu identifiers are defined by the calling application, whereas control notifications are defined by the control.

You may have noticed the opportunity to “pun” the control notification codes. If a control defines a notification code as zero, then it will “look like” a menu item selection, since the high-order word of the `wParam` in the case of a menu item selection is zero. The button control takes advantage of this pun:

```
#define BN_CLICKED          0
```

This means that when the user clicks a button control, the `WM_COMMAND` message that is generated “smells like” a menu selection notification. You probably take advantage of this in your dialog procedure without even realizing it.

(The static control also takes advantage of this pun:

```
#define STN_CLICKED         0
```

but in order for the static control to generate the `STN_CLICKED` notification, you have to set the `SS_NOTIFY` style.)

I stated at the start that the accelerator and menu scenarios are just special cases of the control scenario. If you take the pieces of the `WM_COMMAND` message apart, you’ll see that they fall into two categories:

- What happened? (Notification code.)
- Whom did it happen to? (Control handle and ID.)

In the case of a menu or an accelerator, the “What happened?” is “The user clicked on the menu (0)” or “The user typed the accelerator (1)”. The “Whom did it happen to?” is “This menu ID” or “This accelerator ID”. Since the notification is not coming from a control, the control handle is `NULL` .

I apologize to all you Win32 programmers for whom this is just stating the obvious.

Now that you’re an expert on the `WM_COMMAND` message, perhaps you can solve this person’s problem.

Raymond Chen

Follow