If control-specific messages belong to the WM_USER range, why are messages like BM_SETCHECK in the system message range?

devblogs.microsoft.com/oldnewthing/20070910-00

September 10, 2007

Raymond Chen

When I discussed which message numbers belong to whom, you may have noticed that the messages for edit boxes, buttons, list boxes, combo boxes, scroll bars, and static controls go into the system range even though they are control-specific. How did those messages end up there?

They didn't start out there.

In 16-bit windows, these control-specific messages were in the control-specific message range, as you would expect.

```
#define LB_ADDSTRING      (WM_USER + 1)
#define LB_INSERTSTRING   (WM_USER + 2)
#define LB_DELETESTRING   (WM_USER + 3)
#define LB_RESETCONTENT   (WM_USER + 5)
#define LB_SETSEL         (WM_USER + 6)
#define LB_SETCURSEL      (WM_USER + 7)
#define LB_GETSEL         (WM_USER + 8)
#define LB_GETCURSEL      (WM_USER + 9)
#define LB_GETTEXT        (WM_USER + 10)
...
```

Imagine what would have happened had these message numbers been preserved during the transition to Win32,

(Giving you time to exercise your imagination.)

Here's a hint. Since 16-bit Windows ran all programs in the same address space, programs could do things like this:

```
char buffer[100];
HWND hwndLB = <a list box that belongs to another process>
SendMessage(hwndLB, LB_GETTEXT, 0, (LPARAM)(LPSTR)buffer);
```

This reads the text of an item in a list box that belongs to another process. Since processes ran in the same address space, the address of the buffer in the sending process is valid in the receiving process, so that when the receiving list box copies the result to the buffer, it all works.

Now go back and imagine what would have happened had these message numbers been preserved during the transition to Win32.

(Giving you time to exercise your imagination.)

Consider a 32-bit program that does exactly the same thing that the code fragment above does. The code probably was simply left unchanged when the program was ported from 16-bit to 32-bit code, since it doesn't generate any compiler warnings and therefore does nothing to draw attention to itself as needing special treatment.

But since processes run in separate address spaces in Win32, the program now crashes. Well, more accurately, it crashes *that other program*, since it is the other program that tries to copy the text into the pointer that it was led to believe was a valid buffer but in fact was a pointer into the wrong address space.

Just what you want. A perfectly legitimate program crashes because of somebody else's bug. If you're lucky, the programmers will catch this bug during testing, but how will they know what the problem is, since their program doesn't crash; it's some other program that crashes! If you're not lucky, the bug will slip through testing (for example, it might be in a rarely-executed code path), and the experience of the end user is "Microsoft Word crashes randomly. What a piece of junk." (When in reality, the crash is being caused by some other program entirely.)

To avoid this problem, all the "legacy" messages from the controls built into the window manager were moved into the system message category. That way, when you sent message 0x0189, the window manager knew that it was `LB_GETTEXT` and could do the parameter marshalling for you. If it had been left in the `WM_USER` range, the window manager wouldn't know what to do when it gets message `0x040A` since that might be `LB_GETTEXT`, or it might be `TTM_HITTESTA` or `TBM_SETSEL` or any of a number of other control-specific messages.

Theoretically, this motion needed to be done only for legacy messages; i.e., window messages that existed in 16-bit Windows. (Noting that Windows 95 added some new 16-bit messages, so this remapping had to continue at least through Windows NT 4 with the shell update release.) Nevertheless, the window manager team added the `*_GET*INFO` messages in the system message range even though there was no need to put them there from a compatibility standpoint. My suspicion is that it was done to make things easier for accessibility tools.

Note however that placing new messages in the system message range is more the exception than the rule for the edit box and other “core” controls. For example, the new message `EM_SETCUEBANNER` has the numeric value `0x1501`, which is well into the `WM_USER` range. If you try to send this message across processes without taking the necessary precautions, you will crash the target process.

(Note: Standard disclaimers apply. I won’t bother repeating this disclaimer on future articles.)

Raymond Chen

Follow