WM_CTLCOLOR vs GetFileVersionInfoSize: Just because somebody else screwed up doesn't mean you're allowed to screw up too

devblogs.microsoft.com/oldnewthing/20120913-00

September 13, 2012

Raymond Chen

In a discussion of the now-vestigial `lpdwHandle` parameter to the `GetFileVersionInfoSize` function, Neil asks, "Weren't there sufficient API differences (e.g. WM_CTLCOLOR) between Win16 and Win32 to justify changing the definitions to eliminate the superfluous handle?"

The goal of Win32 was to provide as much backward compatibility with existing 16-bit source code as can be practically achieved. Not all of the changes were successful in achieving this goal, but just because one person fails to meet that goal doesn't mean that everybody else should abandon the goal, too.

The Win32 porting tool PORTTOOL.EXE scanned for things which had changed and inserted comments saying things like

- "No Win32 API equivalent" — these were for the 25 functions which were very tightly coupled to the 16-bit environment, like selector management functions.
- "Replaced by OtherFunction" — these were used for the 38 functions which no longer existed in Win32, but for which corresponding function did exist, but the parameters were different so a simple search-and-replace was not sufficient.
- "Replaced by XYZ system" — these were for functions that used an interface that was completely redesigned: the 16 old sound functions that buzzed your tinny PC speaker being replaced by the new multimedia system, and the 8 profiling functions.
- "This function is now obsolete" — these were for the 16 functions that no longer had any effect, like `GlobalLRUNewest` and `LimitEMSPages`.
- "wParam/lParam repacking" — these were for the 21 messages that packed their parameters differently.
- Special remarks for eight functions whose parameters changed meaning and therefore required special attention.
- A special comment just for window procedures.

If you add it up, you'll see that this makes for a total of 117 breaking changes. And a lot of these changes were in rarely-used parts of Windows like the selector-management stuff, the PC speaker stuff, the profiling stuff, and the serial port functions. The number of breaking changes that affected typical developers was more like a few dozen.

Not bad for a total rewrite of an operating system.

If somebody said, "Hey, you should port to this new operating system. Here's a list of 117 things you need to change," you're far more likely to respond, "Okay, I guess I can do that," than if somebody said, "Here's a list of 3,000 things you need to change." Especially if some of the changes were not absolutely necessary, but were added merely to annoy you. (I would argue that the handling of many GDI functions like `MoveTo` fell into the *added merely to annoy you* category, but at least a simple macro smooths over most of the problems.)

One of the messages that required special treatment was `WM_COMMAND`. In 16-bit Windows, the parameters were as follows:

`WPARAM`	`int nCode`
`LPARAM`	`HWND hwndCtl` (low word)
	`int id` (high word)

Observe that this message violated the rule that handle-sized things go in the `WPARAM`. As a result, this parameter packing method could not be maintained in Win32. If it had been packed as

`WPARAM`	`HWND hwndCtl`
`LPARAM`	`int id` (low word)
	`int nCode` (high word)

then the message would have ported cleanly to Win32. But Win32 handles are 32-bit values, so there's no room for both an `HWND` and an integer in a 32-bit `LPARAM`; as a result, the message had to be repacked in Win32.

The `WM_CTLCOLOR` message was an extra special case of a message that required changes, because it was the only one that changed in a way that required more than just mechanical twiddling of the way the parameters were packaged. Instead, it got split out into several messages, one for each type of control.

In 16-bit Windows, the parameters to the `WM_CTLCOLOR` message were as follows:

`WPARAM`	`HDC hdc`
`LPARAM`	`HWND hwndCtl` (low word)
	`int type` (high word)

The problem with this message was that it had *two* handle-sized values. One of them went into the `WPARAM`, like all good handle-sized parameters, but the second one was forced to share a bunk bed with the type code in the `LPARAM`. This arrangement didn't survive in Win32 because handles expanded to 32-bit values, but unlike `WM_COMMAND`, there was nowhere to put the now-ousted `type`, since both the `WPARAM` and `LPARAM` were full with the two handles. Solution: Encode the type code in the message number. The `WM_CTLCOLOR` message became a collection of messages, all related by the formula

```
WM_CTLCOLORtype = WM_CTLCOLORMSGBOX + CTLCOLOR_type
```

The `WM_CTLCOLOR` message was the bad boy in the compatibility contest, falling pretty badly on its face. (How many metaphors can I mix in one article?)

But just because there's somebody who screwed up doesn't mean that you're allowed to screw up too. If there was a parameter that didn't do anything any more, just declare it a reserved parameter. That way, you didn't have to go onto the "wall of shame" of functions that didn't port cleanly. The `GetFileVersionInfoSize` function kept its vestigial `lpdwHandle` parameter, `WinMain` kept its vestigial `hPrevInstance` parameter, and `CoInitialize` kept its vestigial `lpReserved` parameter.

This also explains why significant effort was made in the 32-bit to 64-bit transition not to make breaking changes just because you can. As much as practical, porting issues were designed in such a way that they could be detected at compile time. Introducing gratuitous changes in behavior makes the porting process harder than it needs to be.

Raymond Chen

Follow