What's the point of letting you change the GCL_CBCLSEXTRA if it has no effect?

devblogs.microsoft.com/oldnewthing/20130814-00

August 14, 2013

Raymond Chen

The documentation for the `SetClassLongPtr` function mentions

> **GCL_CBCLSEXTRA**: Sets the size, in bytes, of the extra memory associated with the class. Setting this value does not change the number of extra bytes already allocated.

What's the point of letting the application change a value if it has no effect?

The `GCL_CBCLSEXTRA` class long grants access to the `cbClsExtra` value that was originally passed in the `WNDCLASS` structure when you called `RegisterClass`, or the Ex-versions *mutatus mutandis*. The intent is for it to be used with `GetClassLong` so you can read the value back, in case you forgot, or if you are inspecting somebody else's class (for example, because you want to superclass it, although `GetClassInfo` is probably a better choice). But since `GetClassLong` and `SetClassLong` take the same class index parameter, once it's defined for one, it's defined for the other.

Okay, well, first, let's explain why it has no effect: The class has already been created. The `cbClsExtra` tells the window manager how much extra memory to allocate in the class when it is created. After the class is created, the value isn't really used any more, but Windows hangs on to the value since it needs to report the value when you call `GetClass-Info`. Trying to change the value is like making changes to a blueprint after the building has finished construction. The blueprints are still on file at the planning office, but changing them has no effect on the building. (Though it will mislead the fire chief who is studying the blueprints in order to decide how to put out the fire that is raging on one of your upper floors.)

Okay, so why does Windows let you change the values if they have no effect?

Let's look at the values of those class longs:

```
#define GCL_MENUNAME        (-8)
#define GCL_HBRBACKGROUND   (-10)
#define GCL_HCURSOR         (-12)
#define GCL_HICON           (-14)
#define GCL_HMODULE         (-16)
#define GCL_CBWNDEXTRA      (-18)
#define GCL_CBCLSEXTRA      (-20)
#define GCL_WNDPROC         (-24)
```

How very strange. They're all even numbers, and negative, too. And the value `-22` is skipped, which lies between `GCL_CBCLSEXTRA` and `GCL_WNDPROC`.

Let's look at what the values were in 16-bit Windows:

```
#define GCL_MENUNAME        (-8)
#define GCW_HBRBACKGROUND   (-10)
#define GCW_HCURSOR         (-12)
#define GCW_HICON           (-14)
#define GCW_HMODULE         (-16)
#define GCW_CBWNDEXTRA      (-18)
#define GCW_CBCLSEXTRA      (-20)
#define GCL_WNDPROC         (-24)
#define GCW_STYLE           (-26)
```

Okay, now it looks even more suspicious. All of the special class values were words (as indicated by the `W` in `GCW`), except for two longs (`GCL`), and the gap exactly falls right where a long would go.

You've probably figured it out by now. In 16-bit Windows, the internal `CLASS` structure looked like this:

```
typedef struct tagCLASS
{
    ... blah blah blah ...
    UINT      style;           // offset -26 from extraBytes
    WNDPROC   lpfnWndProc;     // offset -24 from extraBytes
    int       cbClsExtra;      // offset -20 from extraBytes
    int       cbWndExtra;      // offset -18 from extraBytes
    HMODULE   hModule;         // offset -16 from extraBytes
    HICON     hIcon;           // offset -14 from extraBytes
    HCURSOR   hCursor;         // offset -12 from extraBytes
    HBRUSH    hbrBackground;   // offset -10 from extraBytes
    LPSTR     lpszMenuName;    // offset -8 from extraBytes
    LPSTR     lpszClassName;   // offset -4 from extraBytes
    BYTE      extraBytes[1];   // offset 0 (extra bytes start here)
}
CLASS;
```

When a class was created, the class extra bytes were appended directly to the `CLASS` structure, which meant that you could use negative offsets to access the internal class structures.

```
WORD GetClassWord(HWND hwnd, int index)
{
    CLASS *pcls = GetWindowClassPointer(hwnd);
    WORD *pw = (WORD*)&pcls->cls_extraBytes[index];
    return *pw;
}
LONG GetClassLong(HWND hwnd, int index)
{
    CLASS *pcls = GetWindowClassPointer(hwnd);
    LONG *pl = (LONG*)&pcls->cls_extraBytes[index];
    return *pl;
}
WORD SetClassWord(HWND hwnd, int index, WORD wNewValue)
{
    CLASS *pcls = GetWindowClassPointer(hwnd);
    WORD *pw = (WORD*)&pcls->cls_extraBytes[index];
    WORD wPrevValue = *pw;
    *pw = wNewValue;
    return wPrevValue;
}
LONG SetClassLong(HWND hwnd, int index, LONG lNewValue)
{
    CLASS *pcls = GetWindowClassPointer(hwnd);
    LONG *pl = (LONG*)&pcls->cls_extraBytes[index];
    LONG lPrevValue = *pl;
    *pl = lNewValue;
    return lPrevValue;
}
```

Except of course that the original code was written in assembly language, so it was more like

```
FindClassExtraBytes proc
      mov  bx, [bp][2][4] ;; caller's hwnd
      mov  bx, [bx].wnd_pcls ;; get the class for the window
      add  bx, cls_extraBytes ;; move to extra bytes
      add  bx, [bp][2][4][2] ;; pointer to the requested bytes
      ret
;; use helper macros from cmacros.inc
cProc GetClassWord, <FAR, PUBLIC>
ParmW hwnd
ParmW index
cBegin
      call FindClassExtraBytes
      mov  ax, [bx]      ;; get the word
cEnd
cProc GetClassLong, <FAR, PUBLIC>
ParmW hwnd
ParmW index
cBegin
      call FindClassExtraBytes
      mov  ax, [bx]      ;; get the low word
      mov  dx, [bx][2]   ;; get the high word
cEnd
cProc SetClassWord, <FAR, PUBLIC>
ParmW hwnd
ParmW index
ParmW newValue
cBegin
      call FindClassExtraBytes
      mov  ax, newValue
      xchg ax, [bx]      ;; exchange value
cEnd
cProc SetClassLong, <FAR, PUBLIC>
ParmW hwnd
ParmW index
ParDL newValue
cBegin
      call FindClassExtraBytes
      mov  ax, newValue[0] ;; low word
      mov  dx, newValue[2] ;; high word
      xchg ax, [bx][0]     ;; exchange low word
      xchg dx, [bx][2]     ;; exchange high word
cEnd
```

In other words, the negative offsets were exactly the values needed to access the corresponding fixed fields in the `CLASS` structure as if they were extra bytes. (Again, I marvel at how 16-bit Windows managed to accomplish what it did in so little code. The actual code was even tighter than this.)

There were programs that said, “Hey, since I know I can change this value all I want, and it won’t have any effect, I can use it as a *secret hiding place*,” and instead of storing data in a more sane location, they just squirreled it away in the `GCL_CBCLSEXTRA`.

Windows blocked changes to `GCL_CBCLSEXTRA` starting in Windows 95, but a compatibility loophole was created so that 16-bit programs written for older versions of Windows could still get the old behavior where they could modify a value that had no effect, just so that they could use it as a secret hiding place.

But for all 32-bit programs and newer 16-bit programs, attempting to modify the `cbCls-Extra` value will fail with `ERROR_INVALID_PARAMETER`.

Bonus chatter: Another secret hiding place that applications discovered was storing data in the window extended style bits, `dwExStyle`. “Thanks, Windows, for adding four more bytes of data to each window. I’ll use it to store a pointer! (I’m sure Windows won’t mind.)” There is code in the window manager to enforce the rule that you must use `SetWindowPos` to change the `WS_EX_TOPMOST` style rather than calling `SetWindowLong`, but there is a compatibility loophole: If your application was written for Windows 3.1 and you are setting extended styles that didn’t exist in Windows 3.1, then the window manager says, “I think I know what you’re up to” and suspends the rules so that the application can go ahead and use the extended window style as a secret hiding place.

Raymond Chen

Follow