What went wrong in Windows 95 if you use a system color brush as your background brush?

devblogs.microsoft.com/oldnewthing/20061128-06

November 28, 2006

Raymond Chen

If you want to register a window class and use a system color as its background color, you set the `hbrBackground` member to the desired color, plus one, cast to an `HBRUSH`:

```
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
```

Windows 95 introduced "system color brushes", which are a magic type of brush which always paint in the corresponding system color, even if the system color changes.

```
HBRUSH hbrWindow = GetSysColorBrush(COLOR_WINDOW);
```

The `hbrWindow` brush will always paint in the color corresponding to `GetSysColor(COLOR_WINDOW)`, even if the user changes the color scheme later.

Now, you might be tempted to use a system color brush as your class background brush. After all, if you want the background to be the system window color, why not use a brush that is always the system window color?

Well, because the documentation for `GetSysColorBrush` explicitly tells you not to do that. If you tried this on Windows 95, it would have seemed to work for a while, and then bad things would start happening.

The system color brushes were added as a convenience to save people the trouble of having to create solid color brushes just to draw a system color. Profiling in Windows 95 revealed that a lot of time was spent by applications creating solid brushes, using them briefly, and then destroying them, so anything that could be done to reduce the rate at which applications needed to do this was a good thing.

The system color brushes were implemented in Windows 95 by creating them and then telling GDI, "Hey, if somebody tries to `DeleteObject` this brush, don't let them." This prevented the system color brushes from being accidentally destroyed.

Except when it didn't.

When you registered a window class with a background brush (and by that, I mean an honest-to-goodness brush and not a pseudo-brush you get from that `(HBRUSH)(COLOR_xxx + 1)` stuff) the window manager did the same thing to the brush as it did to the system color brushes: It told GDI, "Hey, if somebody tries to `DeleteObject` this brush, don't let them." This prevented people from destroying brushes while the window manager was still using them to draw the background of a window.

When you unregistered the window class, the window manager told GDI, "Okay, delete this brush, and yes, I told you not to let anyone `DeleteObject` it, but I'm overriding it because I was the one who protected it in the first place. I'm just cancelling my previous instructions to protect this brush." The window manager takes responsibility for destroying the brush when you register the class; therefore, when you unregister the class, the window manager is obligated to clean up and destroy the brush. (Actually, it's a little more complicated than that, because it is legal to use one brush as the background brush for multiple window classes, but let's ignore that case for now.)

Do you see the problem yet?

What if you registered a window class with a system color brush as the background brush and then unregistered it? (Don't forget that classes are automatically unregistered when the process exits.) When you registered the class, the brush got protected, and when you unregistered the class, the Windows 95 window manager **told GDI to override the protection and destroy the brush anyway**.

Oops, we just destroyed a system color brush. Even though those brushes were protected, the protection didn't work here because you went through the code path where the window manager said, "Override the safety interlocks!"

But of course, you didn't need to use a system color brush in the first place. You should have used that `(HBRUSH)(COLOR_xxx + 1)` pseudo-brush.

(Note to nitpickers: This story talked about Windows 95. It does not apply to any other version of Windows. The problem may or may not exist in those other versions; I make no claims.)

Raymond Chen

Follow