How is LockWindowUpdate meant to be used?

devblogs.microsoft.com/oldnewthing/20070220-07

February 20, 2007

Raymond Chen

Now that we know how `LockWindowUpdate` works, we can look at what it is for. Actually, the intended purpose of `LockWindowUpdate` can be captured in one word: dragging. But we'll get to that a little later. The purpose of `LockWindowUpdate` is to allow a program to **temporarily take over the responsibility of drawing a window**. Of course, in order to do this, you have to prevent the window procedure (or anybody else) from doing their normal drawing activities; otherwise, the two pieces of code (the code that normally draws the window and the code that's trying to take over) fight for control of the window and you get an ugly mess since neither knows what the other is doing. But if you've locked the window for updating, how do **you** draw to it? You use the `DCX_LOCKWINDOWUPDATE` flag to the `GetDCEx` function. This flag means, "Let me draw to the window even though it is locked for update." Only the code that locked the window for update should pass this flag, of course, since it would otherwise create the conflict that `LockWindowUpdate` was intended to avoid in the first place. Since people like tables so much, I've made one that summarizes what changes when a window is locked for update:

	Normal behavior	**Window locked for update**
`BeginPaint`, `GetDC`, ***etc.***	Drawing operations paint the window	Drawing operations paint nothing, but the affected area is remembered for future invalidation
`GetDCEx` **with** `DCX_LOCKWINDOWUPDATE` **flag**	(do not use)	Drawing operations paint the window

In other words, when a window is locked for update, the ability to draw to the window is taken away from the normal DC-acquisition functions (`BeginPaint` and friends) and given to `GetDC(DCX_LOCKWINDOWUPDATE)` . Note that if no window is locked for update, you should not pass the `DCX_LOCKWINDOWUPDATE` flag; the purpose of that flag is to indicate "I'm the guy who called `LockWindowUpdate` . Let me in!" It's sort of the window manager equivalent of the old comedy routine where you tell the guard, "Nobody is allowed into this

room." And then you come back an hour later and the guard won't let you in. "I'm sorry, sir, but I'm not allowed to let anyone into this room." "But I'm the one who told you not to let anyone into the room." "That's right sir, and I'm following your orders. Nobody is allowed into the room." The mistake was in the initial order to the guard. You should have said, "Nobody except me is allowed into this room." And `DCX_LOCKWINDOWUPDATE` is how you tell the window manager, "It's me. Let me in." If you go back and look at the way the `LockWindowUpdate` function works, you'll see that if the window that was locked doesn't try to draw, then when the window is unlocked, nothing is invalidated. Whereas the `CS_SAVEBITS` class style automatically saves the original pixels and restores them when the window is removed from the screen, the `LockWindowUpdate` doesn't do any such thing. It is your responsibility to ensure that any pixels you changed while the window was locked for update have been restored to their original values when you call `LockWindowUpdate(NULL)`. This is typically done by saving the original pixels into an off-screen bitmap before you do your custom painting and putting them back when you're done. Okay, so here's the intended usage pattern:

- When you want to take over drawing from another window, call `LockWindowUpdate` on that window.
- Save the pixels from that window that you're going to overwrite.
- Draw your new pixels. (These pixels are often a modification of the original pixels. For example, you might add the image of an object that is being dragged over that window.)
- Repeat as necessary as long as your operation is still in progress. (Doing so may require you to "back up" more pixels of the screen if the region of the screen you're modifying is different from the region you modified originally. You can do this backup/restore incrementally. For example, instead of accumulating the set of pixels you need to restore, you can restore all the original pixels, compute the new position of the drag image, save those new pixels, and draw the new image. That way, you have only one set of "backup pixels" to deal with.)
- When the operation is complete, restore the original pixels and call `LockWindowUpdate(NULL)`.

Next time, we'll look more at that one word "dragging" and how it is closely tied to the whole concept of `LockWindowUpdate`. Even though we've only started talking about `LockWindowUpdate`, you should already know enough to answer this question.

(Note: The purpose of this series is to describe the way the `LockWindowUpdate` was intended to be used, not to discuss whether it was a good design in the first place.)

Raymond Chen

Follow