The consequences of invalidating the null window

devblogs.microsoft.com/oldnewthing/20060307-08

March 7, 2006

Raymond Chen

On occasion, you might notice that every window on the desktop flickers and repaints itself. One of the causes for this is a simple null handle bug. The `InvalidateRect` function is one you're probably well-familiar with. It is used to indicate to the window manager that the pixels of a particular window are no longer current and should be repainted. (You can optionally pass a rectangle that specifies a subset of the window's client area that you wish to mark invalid.) This is typically done when the state of the data underlying the window has changed and you want the window to repaint with the new data. If however you end up passing `NULL` as the window handle to the `InvalidateRect` function, this is treated as a special case for compatibility with early versions of Windows: It invalidates **all** the windows on the desktop and repaints them. Consequently, if you, say, try to invalidate a window but get your error checking or timing wrong and end up passing `NULL` by mistake, the result will be that the entire screen flickers.

Even more strangely, passing `NULL` as the first parameter `ValidateRect` has the same behavior of **invalidating** all the windows. (Yes, it's the "Validate" function, yet it invalidates.) This wacko behavior exists for the same compatibility reason. Yet another example of how programs rely on bugs or undocumented behavior, in this case, the peculiar way a `NULL` parameter was treated by very early versions of Windows due to lax parameter validation. Changing nearly **anything** in the window manager raises a strong probability that there will be many programs that were relying on the old behavior, perhaps entirely by accident, and breaking those programs means an angry phone call from a major corporation because their factory control software stopped working.

Raymond Chen

Follow