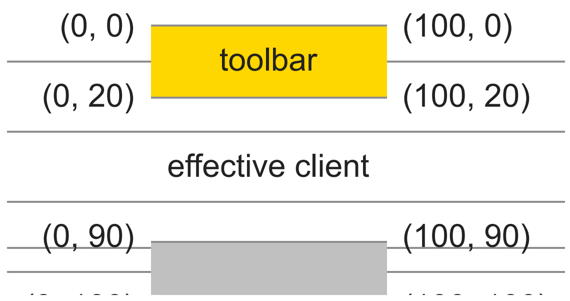
The forgotten common controls: The GetEffectiveClientRect function

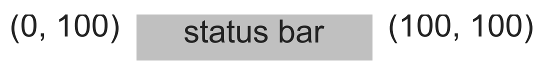
devblogs.microsoft.com/oldnewthing/20071123-00

November 23, 2007

Raymond Chen

The `GetEffectiveClientRect` function is another one in the category of functions that everybody tries to pretend doesn't exist. It's not as bad as `MenuHelp`, but it's still pretty awful. The idea behind the `GetEffectiveClientRect` function is that you have a frame window with a bunch of optional gadgets, such as a status bar or toolbar. The important thing is that these optional gadgets all reside at the borders of the window. In our examples, the toolbar goes at the top and the status bar goes at the bottom. You might also have gadgets on the left and right such as a navigation tree or a preview pane. They can also be stacked up against the border, such as an address bar and a toolbar. The important thing is that all the gadgets go around the border. The first parameter to the `GetEffectiveClientRect` function is the window whose effective client rectangle you wish to compute; no surprises there. The second parameter is a pointer to the rectangle that receives the result; again, hardly surprising. It's that third parameter, the array of integers, that is the weird one. The first two integers in the array are ignored. The remainder of the array consists of pairs of nonzero integers; the array is terminated by a pair consisting of zeroes. Of each pair, only the second integer is used; it is the control identifier of a child window of the window you passed in. If that child window is visible (in a special sense I'll explain later), then its window rectangle is subtracted from the parent window's client rectangle. After all the rectangles of visible children are subtracted away, what remains is the *effective client rectangle*. For example, suppose your window's client rectangle is 100×100 and there is a toolbar at (0, 0)–(100, 20) and a status bar at (0, 90)–(100, 100), both visible. The `GetEffectiveClientRect` starts with the full client rectangle (0, 0)–(100, 100), subtracts the two rectangles corresponding to the toolbar and status bar, resulting in (0, 20)–(100, 90).





If the control IDs for the toolbar and status bar are 100 and 101, respectively, then the array you need to pass would be `{ *, *, ¤, 100, ¤, 101, 0, 0 }` where * can be anything and ¤ can be any nonzero value. Continuing from the above example, if the status bar were hidden, then the effective client rectangle would be (0, 20)–(100, 100) because hidden windows are ignored when computing the effective client rectangle. Okay, first question: What is that special sense of visible I mentioned above? I didn't write simply *visible* because `IsWindowVisible` reports a window as visible only if the window *and all its parents* are visible. But all that `GetEffectiveClientRect` cares about is whether the window is visible in the sense that the `WS_VISIBLE` style is set. In other words, that the window *would be* visible if its parent is. Why does the `GetEffectiveClientRect` use this strange definition of visible? Because it wants to make it possible for you to get the effective client rectangle of a window while it is still hidden, the result being the effective client rectangle you would get once the window becomes visible. This is valuable because it allows you to do your calculations "behind the scenes" while the window is still hidden (for example, in your `WM_CREATE` handler). Second question: Why is the integer array so crazy? What's with all the ignored values and the "must be nonzero" values? Why can't it just be the array `{ 100, 101, 0 }`? The format of the integer array is the same as the one used by the `ShowHideMenuCtl` function. The intent was that you could use the same array for both functions. The two functions do work well together: The `ShowHideMenuCtl` function do the work of letting the user toggle the toolbar and status bar on and off, and `GetEffectiveClientRect` lets you compute the client rectangle that results.

That said, the `GetEffectiveClientRect` function is largely ignored nowadays. It doesn't do anything you couldn't already do yourself, and when you write your own version, you don't need to deal with that crazy integer array.

Raymond Chen

Follow