Why does the primary monitor have (0,0) as its upper left coordinate?

devblogs.microsoft.com/oldnewthing/20100820-00

August 20, 2010

Raymond Chen

By definition, the primary monitor is the monitor that has (0,0) as its upper left corner. Why can't the primary monitor be positioned somewhere else? Well, sure you could do that, but then you'd have to invent a new name for the monitor whose upper left corner is at (0,0), and then you're back where you started. In other words, it's just a name. You could ask, "Why can't starboard be on the left-hand side of the boat?" Well, sure you could do that, but then you'd have to come up with a new name for the right-hand side of the boat, and then things are pretty much the same as they were, just with different names. Perhaps a more descriptive (but clumsier) name for the primary monitor would be the *backward-compatibility monitor (for applications which do not support multiple monitors)*, because that's what the primary monitor is. If an application is not multiple-monitor aware, then any time it asks for properties of "the" monitor, it gets information about the backward-compatibility monitor. A call to `GetSystemMetrics(SM_CXSCREEN)` gives the width of the backward-compatibility monitor, `GetSystemMetrics(SM_CYMAXIMIZED)` gives the height of a window maximized on the backward-compatibility monitor, and positioning a window at (0,0) puts it at the upper left corner of the backward-compatibility monitor. The window manager still respects window coordinates passed to functions like `CreateWindow` or `SetWindowPos`. If you pass coordinates that are on a secondary monitor—oops—a monitor different from the backward-compatibility monitor, then the window manager will happily put the window there. These coordinates might be the result of a program that is multiple-monitor aware, or possibly merely from a program which is multiple-monitor agnostic. Multiple-monitor agnosticism is a term I just made up which refers to programs which might not explicitly support multiple monitors, but at least were open to the possibility of multiple monitors by not making assumptions about the number of monitors but instead using functions like `RectVisible` to determine what the visible portions of the screen are. These techniques were hot topics many years ago when you wanted to write a program that ran both on single-monitor-only versions of Windows as well as multiple-monitor versions of windows; nowadays there are rather old-fashioned, like coming up with mnemonics for all your friends' telephone numbers so you didn't have to keep looking them up. (Today, you just go ahead and call the multiple monitor functions and use the address book function in your mobile phone to remember your friends' phone numbers.) It is not the case that the primary monitor is the *applications*

show up here first monitor. As noted earlier, applications show up on whatever monitor they ask for, whether they asked for it explicitly (hunting around for a monitor and using it) or implicitly (restoring to the same coordinates they were on when they were last run).

Of course, programs which pass `CW_USEDEFAULT` to the `CreateWindow` function explicitly abdicated the choice of the window position and therefore the monitor. In that case, the window manager tries to guess an appropriate monitor. If the new window has a parent or owner, then it is placed on the same monitor as that parent or owner; otherwise, the window manager just puts the window on the backward-compatible monitor, for lack of a better idea.

Raymond Chen

Follow