How can I update my WinForms app to behave better at high DPI, or at normal DPI on very large screens?

devblogs.microsoft.com/oldnewthing/20160617-00

June 17, 2016

Raymond Chen

A customer had a WinForms app that was showing its age. It didn't scale itself properly on high-DPI screens, which means that if you ran it on a Surface Pro 4, you got a teeny-tiny window and then you had to go grab your magnifying glass in order to read it. The customer noted, "When the program is run on a large display, it occupies only a tiny portion of the screen. What is the least amount of work I need to do to get this program to look less awful on large screen? I am willing to accept blurriness in exchange for doing less work."

The lowest level of DPI awareness is "none", which is what you get if you do not manifest your application, you have a manifest but do not declare whether you are DPI aware, or if you have a manifest and you declare your DPI awareness as `<dpiAware>False</dpiAware>`. (The value is not case-sensitive.) For an application that is not DPI-aware, the operating system emulates a display that is 96 DPI: If the program asks for the monitor's DPI, it will be told that the monitor is 96 DPI. If the program asks for the resolution of the screen, the operating system adjusts the number so that the program sees a monitor that is filled with 96 DPI pixels. What actually happens is that the program's output is scaled so that each pixel is one 96th of an inch square. And the result is a little blurry because that's the nature of scaling up.

The next level of DPI awareness is "system DPI awareness", which is what you get if you manifest your application as `<dpiAware>True</dpiAware>`. For these applications, the operating reports the actual DPI of the highest-DPI monitor as the DPI of all monitors. If the application displays a window on the highest-DPI monitor, it is shown at actual size. If the application displays a window on any other monitor, it is scaled down as necessary so that each pixel is one Xth of an inch square, where X is the DPI of the highest-DPI monitor. (Since the highest DPI was reported to the app, the application's out will either be unscaled or scaled down. Scaling down results in less blurry results than scaling up, which is why the operating system reports the highest DPI of any monitor.)

The third level of DPI awareness is "per-monitor DPI awareness", which you opt into by setting `<dpiAware>True/PM</dpiAware>`. This strange not-really-a-Boolean value is a trick that takes advantage of the fact that versions of Windows prior to Windows 8.1 checked only

that the value of `dpiAware` began with the letters t-r-u-e and ignored any trailing junk. As a result, a value of `True/PM` is interpreted as `True` on older systems, which means that an application that uses this manifest declaration gets per-monitor DPI on Windows 8.1 and higher, or system DPI on Windows 8 and lower.

Given the customer's willingness to accept blurriness in exchange for doing less work, the thing to do is to mark the program as DPI-unaware by setting `<dpiAware>False</dpiAware>` in the manifest, or leaving it out entirely.

The customer reported that they didn't have any such declaration in their application's manifest, but they were being treated as system DPI aware. Some investigation revealed that the very first line in its `Main` function is a call to the `SetProcessDpiAwareness` function. That call is equivalent to setting the DPI awareness in a manifest, but you have to call it before you do anything that is dependent upon DPI.

Okay, so problem solved, right? Remove the call to `SetProcessDpiAwareness` from the `Main` function, and now the program will be treated as DPI-unaware, and it will render at 96 DPI and be scaled up (blurrily) on higher-DPI monitors.

But of course that's not the entire problem. The customer explained that this got rid of the teeny-tiny text, but that wasn't the entire problem they had. "When users run the program on a machine with a very large screen, the window is properly scaled, but it is not taking advantage of the fact that the screen is a large 24-inch monitor. The program uses only about a third of the screen."

Okay, so the deal here is not that the program is rendering text that is too small to be read. The text is a perfectly readable size. The issue is that the developer wants the program to resize itself to something that covers more of the screen. That's not a DPI issue; that's just a program changing its default window size based on the screen dimensions. When your program starts up, check the dimensions of the monitor it is displayed on, do your calculations, and resize yourself if your calculations say that you want to be bigger.

Bonus reading:

Raymond Chen

Follow