Why are console windows limited to Lucida Console and raster fonts?

devblogs.microsoft.com/oldnewthing/20070516-00

May 16, 2007

Raymond Chen

In Windows 95, we experimented with other fonts for the console window, and it was a disaster.

In order to be a usable font for the console window, the font needs to be more than merely monospace. It also needs to support all the characters in the OEM code page. Testing this is easy for SBCS code pages, since they have only 256 characters. But for DBCS code pages, testing all the characters means testing tens of thousands of code points. The OEM code page test already rules out a lot of fonts, because the 437 code page (default in the United States) contains oddball characters like the box-drawing characters and a few astronomical symbols which most Windows fonts don't bother to include.

But checking whether the font supports all the necessary characters is a red herring. The most common reason why a font ends up unsuitable for use in a console window is that the font contains characters with negative A- or C-widths. These A- and C-width values come from the ABC structure and represent the amount of under- and overhang a character consumes.

Consider, for example, the capital letter W. In many fonts, this character contains both under- and overhang:

X						X						X
X					X		X					X
	X				X		X				X	
	X				X		X				X	
	X				X		X				X	
		X		X				X		X		

	X	X			X	X	
	X	X			X	X	
		X				X	
		X				X	

Notice how the left and right stems “stick out” beyond the putative cell boundaries.

I wrote code in Windows 95 to allow any monospace font to be used in console windows, and the ink was hardly even dry on the CD before the bugs started pouring in. “When I choose Courier New as my font, my console window looks like a Jackson Pollock painting with splotches of pixels everywhere, and parts of other characters get cut off.” (Except that they didn’t use words as nice as “splotches of pixels”.)

The reason is those overhang pixels. The console rendering model assumes each character fits neatly inside its fixed-sized cell. When a new character is written to a cell, the old cell is overprinted with the new character, but if the old character has overhang or underhang, those extra pixels are left behind since they “spilled over” the required cell and infected neighbor cells. Similarly, if a neighboring character “spilled over”, those “spillover pixels” would get erased.

The set of fonts that could be used in the console window was trimmed to the fonts that were tested and known to work acceptably in console windows. For English systems, this brought us down to Lucida Console and Terminal.

“Why isn’t there an interface for choosing a replacement font, with a big annoying message box warning you that ‘Choosing a font not on the list above may result in really ugly results. Don’t blame me!’?”

First of all, because we know that nobody reads those warnings anyway. Second, because a poor choice of font results in the console window looking so ugly that everybody would rightly claim that it was a bug.

“No, it’s not a bug. You brought this upon yourself by choosing a font that results in painting artifacts when used in a console window.”

“Well, that’s stupid. You should’ve stopped me from choosing a font that so clearly results in nonsense.”

And that’s what we did.

Of course, if you're a super-geek and are willing to shoulder the blame if the font you pick happens not to be suitable for use in a console window, you can follow the instructions in this Knowledge Base article to add your font to the list. But if you end up creating a work of modern art, well, you asked for it.

Nitpicker's corner

In the title of this entry, s/console windows/Windows console windows/†

†s/Windows console windows/Windows console windows when displayed inside a GUI window, as opposed to consoles that have gone to hardware fullscreen, which is another matter entirely/.

Raymond Chen

Follow