Whether the Unicode Bidi algorithm is intuitive depends on your definition of "intuitive"

devblogs.microsoft.com/oldnewthing/20121026-00

October 26, 2012

Raymond Chen

In Windows, we spend a good amount of time with the pseudo-mirrored build. And one of the things that you notice is that pseudo-mirrored text comes out looking really weird. For example, the string really? (yup). comes out pseudo-mirrored as .(really? (yup. Just for fun, here's here's how your browser renders it:

> really? (yup).

Even stranger, the IPv6 address 2001:db8:85a3::8a2e:370:7334 comes out as db8:85a3::8a2e:370:7334:2001. (The IPv6 address was the string that prompted this article.) The result of the RTL IPv6 address is even weirder if you force a line break at a particular point. If your browser follows the Unicode Bidi algorithm, you can resize the box below to see how the line break position affects the rendering.

> 2001:db8:85a3::8a2e:370:7334

If your browser doesn't follow the Unicode Bidi algorithm, or if you can't resize the window, here's what you get:

No line break	db8:85a3::8a2e:370:7334:2001
Line break	:2001 db8:85a3::8a2e:370:7334

"Is this a bug?" No. Well, maybe yes. It depends. But mostly yes. Windows is following the Unicode Bidirectional Algorithm. So the part that's not a bug is "Windows is correctly following an international standard." The weirdness you're seeing is just a consequence of following the standard. Let's look at what's going on. When you render text in RTL context, what you're saying is "Render this text in the form you would see it if it appeared in a newspaper printed in an RTL language." For illustration, we follow the convention that uppercase characters are considered to be in an RTL script, lowercase characters are considered to be in an LTR script, and non-letters stand for themselves. Say you want to

render the string “NEXT COMES john smith.” A newspaper would say, “Well, my readership expects things to be laid out right to left. The string ‘john smith’ is a foreign name inserted into a paragraph that otherwise is written my readers’ native language. If the name were in my readers’ native language, I would render it as

> .HTIMS NHOJ SEMOC TXEN

Since the name is in a foreign language, I will treat it as an opaque ‘name blob’ that got inserted into my otherwise beautiful RTL sentence.”

> .john smith SEMOC TXEN

(The black outline is not part of the actual output. I am using it to highlight that the phrase *john smith* is being treated as a single unit.) This also explains why “hello.” comes out as “.hello“. The LTR text is treated as a blob inside an otherwise RTL sentence.

> .hello

Things get weirder once parentheses and digits and more complex punctuation marks are thrown into the mix. For example, the Unicode Bidirectional Algorithm has to figure out that in the text “IT IS A bmw 500, OK.” the “500” is attached to the LTR text “bmw”, resulting in

> .KO ,bmw 500 A SI TI

And it also needs to work out the correct text rendering order when you have RTL text embedded inside LTR text, all of which is embedded inside other RTL text, as illustrated by the brain-teaser “DID YOU SAY ’he said “car MEANS CAR”‘?” But maybe the standard is buggy. The problem is that the Unicode Bidirectional Algorithm is designed for text, so when you ask it to render things that aren’t text (such as IPv6 addresses and URLs), the results can be nonsensical. At least for the IPv6 case, you can work around the problem by explicitly marking the IPv6 address as LTR, so that the Unicode Bidirectional Algorithm doesn’t get involved, and the characters are rendered left-to-right in the order they were written.
Exercise: Study the Unicode Bidirectional Algorithm and explain why really? (yup). comes out as .(really? (yup.

Bonus reading: What you need to know about the bidi algorithm and inline markup.

Raymond Chen

Follow