How can CharUpper and CharLower guarantee that the uppercase version of a string is the same length as the lowercase version?

devblogs.microsoft.com/oldnewthing/20200804-00

August 4, 2020

Raymond Chen

The `CharUpper` function takes a buffer of characters and converts them in place to uppercase. This requires that the uppercase version of any character occupy the same number of code units as the lowercase counterpart. However, there is nothing in the Unicode specification that appears to require this. Did Microsoft come to some sort of special under-the-table deal with the Unicode Consortium to ensure that this property holds for all characters?

No, there is no such special under-the-table deal, probably because there is also no such guarantee. And in fact, there are counterexamples if you look closely enough. We noted earlier that the uppercase version of the ß character for a long time was the two-character combination SS. (This got even more complicated with the adoption of the capital ẞ in 2017.) There's also U+1F80 GREEK SMALL LETTER ALPHA WITH PSILI AND YPOGEGRAMMENI "ᾀ" whose uppercase version is the two characters U+1F08 GREEK CAPITAL LETTER ALPHA WITH PSILI and U+0399 GREEK CAPITAL LETTER IOTA "ἈΙ". But if you ask `CharUpper` to convert them, it leaves ß unchanged, and it converts "ᾀ" to U+1F88 GREEK CAPITAL LETTER ALPHA WITH PSILI AND PROSGEGRAMMENI "ᾈ".

The `CharUpper` function tries to convert the string in place, but if the uppercase and lowercase versions of a character are not the same length, then it panics and does something strange.

The `CharUpper` function is a legacy function that remains for compatibility with the `AnsiUpper` function in 16-bit Windows, and we saw last time that the `AnsiUpper` function was originally hard-coded to code page 1252. Over time, Windows added support for other code pages, and they happened to have enjoyed the property that the uppercase and lowercase versions of a string have the same length. (Again, if you ignore the weird ß ↔ SS thing.)

Eventually, that rule broke down, but you can't go back in time and kill `CharUpper`'s parents before it was born. You just have to accept that there's this thing called `CharUpper` that has some baked-in assumptions that are wrong. If you give it a string that violates those assumptions, then it does what it can, but the results aren't the best.

I would consider the `CharUpper` and `CharLower` family of functions to be deprecated. Instead, use the `LCMapStringEx` function with the `LCMAP_UPPERCASE` or `LCMAP_LOWERCASE` flag, as appropriate.

Raymond Chen

Follow