Subject: Re: Choosing the best way to go full UNICODE
Posted by mirek on Sun, 11 Jun 2017 11:57:38 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 08 June 2017 13:00Not quite 100% done analyzing the data, but here is what I think I'll do:
- respect the 3 plane convention. Unicode has 17 planes, with the first 3 in active use. Plane 14 is used, but it is specific and only has 368 allocated code points. It is so specific that I'll add exclude it, the same as I do all planes except planes 0-2. All excluded planes have the property that any function f(cp) = cp.
- I'll ignore all special substitutions: sub and superscript, font, circle, square, fractions and of course compatibility substitutions. I won't be using a flag for now, just exclude them.
- I'll ignore all CJK COMPATIBILITY IDEOGRAPHs. There is no way a general purpose library can provide satisfactory use case for these. If you really needs such substitution, you will probably use a more competent third party library. f(CJK COMPATIBILITY IDEOGRAPH) = CJK COMPATIBILITY IDEOGRAPH

All these combined with my two table solution, with a chunk size of 256 to 1024 will leave me with around 8000-9000 bytes of data in each executable that does decomposition. Final numbers will be determined once implementation is done and round trip testing is complete.

I think this is a reasonable subset that can handle NFD, at the small price of a flat 9K in exe size, + the size of the actual methods.

I have managed to squeeze complete composition to 3.8KB table... :)

Interesting observation: With UnicodeCompose / Decompose, with first 2048 codepoints covered by "fast table", there is no need for further tables for ToUpper, ToLower, ToAscii.