Subject: Re: 16 bits wchar
Posted by mirek on Tue, 05 Aug 2008 13:12:31 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 05 August 2008 06:03luzr wrote on Tue, 05 August 2008 01:51
Anyway, might I ask you to think about / comment codepoint == glyph and distinct(codepoint) < 64K claims?

I really can't imagine how that would be possible.

First of all, how do you expect to squish almost 100K characters in 64K? Some kind of dynamic character set loading would be needed, and still a string could not contain every possible character.

Yes, meanwhile I have studied it a little bit more, you are right.

Quote:
And second, in Unicode codepoint != glyph. All the 90k+ codepoints can be combined theoretically to produce and endless number of glyphs. Think of Unicode as a comparably more feature poor Qtf. Codepoints are commands. 99% of commands are "print glyph X", but the rest allow you to manipulate the layout and appearance of glyph. It is not a visual manipulation, like with font, rather manipulation that alters the abstract concept of a glyph, like adding diacritics.

Well, I was studying this as well and came to conclusion that combining is of little concern.

First, AFAIK, basic Unicode "compliance" does not requite it.

Second, all important ("real") combining codepoints have characters in Unicode.

IMO, I would regard combining as sort of formating info, similar to '\n' or '\t' - something that we need to be aware about (and, in fact, we already are, sort of, see UnicodeCombine...) but do not need to actively support in editors etc...

BTW, that UnicodeCombine is exactly the sort of support that makes sense.

Quote:
All common diacritics are handled pretty well, but uncommon ones which are often incorrectly handled.

This is because there is no general way how to create combined glyph....

Quote:
Under Windows, when you use such text in non editable controls in U++, you get correct result, but if you use an EditString for example, you have to press cursor keys multiple times to step through a character which visually is made out of only one glyph, but uses several code points as

representation.

Does not make sense to me...

Quote:
This problem can be relatively easily addressed, by updating a couple of functions and making sure than Windows API always gets full chunks of text.

IMO, this would be pretty hard to address in fact. Or result in confusing user interface.

Quote:
Under Linux, such support is a lot poorer. Since we send to X text one codepoint at a time, no composition can take place.

Actually, we do not. Interface accepts strings. But I doubt it manages combining.

Quote:
And I don't even know if the methods from X that are in use can handle such texts. All my experiments in U++ gave the same result: diacritics are removed and the rest of characters are displayed as whitespace. KDE editors seemed quite happy with such codes, while gedit displayed the characters correctly, but without composing them in the same place., so basically it did not do any better than U++ if we would have font pooling.

We will, I promise (Well, I would rather describe it as "font substitution"...).

Quote:
As always, I come to the same conclusion: nobody really cares for proper internationalization and Unicode

The question is how combining really helps... IMO, it is not worth the enormous trouble it brings...

Quote:
Yes, that would help under windows and is must under Linux. We could even use some "heuristics", i.e. if a font has 2 Arabic characters, there is a high probability that it handles all Arabic characters from that given Unicode range. Maybe we can get away by splinting all codepoints into ranges on a per script basis, and only test some key characters, but I can't be sure without testing.

Oh, for the beginning, I was rather thinking about "offline experimental scan" to find out what is really going on

Maybe we should then match "standard substitution fonts" for all basic fonts.

Also interesting point is what then happens to my heurestic "glyph fixing" for characters 256-512 (U++ synthetises missing glyphs there by combining characters 0-256). So it is sort of alternative approach to font substitution. But I would keep it as it results in better looking texts.

Mirek
