Subject: Re: 16 bits wchar
Posted by cbpporter on Tue, 05 Aug 2008 10:03:04 GMT
View Forum Message <> Reply to Message

luzr 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.

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.

The reason why this is not that obvious is that Win API handles this for you automatically. Most users and even developers are not familiar with this process, and if somehow their input data contains such characters, Win controls will display them correctly. All common diacritics are handled pretty well, but uncommon ones which are often incorrectly handled. This could be one of U++ strong points in the future. When all font issues are resolved (probably not before 2009.1), if we would offer full combining characters support algorithmically where fonts fail, we would certainly be in a relatively unique position.

but since we don't use native controls, we are more exposed to them. 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.

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.

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

As always, I come to the same conclusion: nobody really cares for proper internationalization and Unicode (except Qt or KDE, who seems to have best support out of all, comparable and maybe better than Windows, but seemingly poorer because of available fonts).

Quote:

Hm, I was thinking about our problem a lot....

I believe that we should do one important thing first - scan all available fonts and count/list all codepoints there...

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.
