Subject: Re: 16 bits wchar
Posted by cbpporter on Mon, 04 Aug 2008 20:47:30 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 04 August 2008 18:14
Well, this rather sound like we should kick out WString altogether and keep just UTF-8:)

I think that we should keep both, and even add LString eventually just for the sake of completeness. In other package if your worried about exe size.

Quote:
Maybe we just need smarter encoding than UNICODE?

Makes me think - realistically, there is a lot of "reserved" positions in BMP. Could we just use them for this?

Well there is nothing better than Unicode AFAIK. It may seem sometimes like there is too much fuss with it, but if you are in my place and have to deal with other legacy encodings, you would have to deal with EUC, EUC-JP, ShiftJIS, JIS and a couple of ISOs, where a lot of these encoding don't guarantee round-trip conversion, and you'll see that Unicode is a true blessing. Great that I have iConv to ease the burden a little.

And BTW, Unicode forbids the use of the reserved or unassigned code points for any use .

Anyway I ran my benchmarks on my Windows machine where console output still works. I did the tests with some experimental methods which are not complete, so the results could be a little inaccurate, but they are still interesting enough too post.

I used 3 methods to convert from a two UTF8 sets to UTF16. The first method is the standard U++ FromUtf8. The second is my FromUtf8SR, which takes into account 4 byte characters, and the third is the highly experimental FromUtf8SR2. The first data set consists of 200 latin characters, representing 200 code points (the letter c 200 times). The second one consists of 100 kanji, 3 characters each, totaling 300 bytes. On second thought, I should have used same sized data sets. All conversions are run 1000000 times.

In Debug mode:
Latin
3125
3203
2078
Kanji
3891
3906
2406

Nothing too impressive here. First method, the standard one is a little faster than mine, and the experimental one is considerably faster.

In Release mode:
Latin
484
485
390
Kanji
4718
3157
812

Here, for kanji, my method really is a lot faster. But in release mode, FromUtf8 for an all kanji input is slower than in Debug mode. Can someone verify this? Maybe I messed something up.

As I said my experimental method is really experimental and not complete yet (I hope it is thread safe also). I hope I'm not chasing after wild geese (is that an expression?) and I didn't miss something that should render my experimental method useless or wrong, because the numbers are great!