Subject: Re: 16 bits wchar
Posted by cbpporter on Thu, 04 Oct 2007 17:49:18 GMT
View Forum Message <> Reply to Message

luzr wrote on Thu, 04 October 2007 17:33OK, patch applied. And you are right about 0xC2, I missed the fact that 0xC0 and 0xC1 is represented by single byte...

Mirek
Did you replace the other one or do you plan to support both versions of Unicode? (5 - mine and what you have - I think 3 or 4). Hope there is no code that depends on six byte Utf-8, but I doubt that this will be an issue for U++.

I will tell you a little about what I'm implementing next. Right now you have a system which allows the use of ill-formatted Utf-8. When transmitted to GUI, it is converted to a valid Utf-16, and if needed you can convert it back to the same Utf-8. This system works, but it kind of creates a bias toward Utf-16. I know that there are objective reasons for this, and Utf-16 is the best choice for Win and a reasonable for other systems, but I would like to be able to process all Unicode formats without regard to OS interaction, efficiency and other issues. If I want to write and i18n GUI application, I'll use WString. If I want to write a console app which specializes in Utf8 or Utf32, I can process those in their native format without need for conversions.

In order to do this, I need to Utf-8 that is corrected by conversion will no longer suffice. The error escaping must be done directly on the Utf-8 and this way there will be no need to error escape at conversions, only at load and save.

This way the normal methods will remain the same. For example, you could still use FromUtf8(in.GetLine()) and all your methods without modification. If you want to do special Utf processing (not needed in normal apps), you will use a new API which takes a "raw" Utf-8 string and escapes it if needed with something like:

```
String  ToUtf8(char code);
String  ToUtf8(const char *s, int len);
String  ToUtf8(const char *s);
String  ToUtf8(const String& w);
```

or other name to not create confusion with wide char variants.

You will use something like ToUtf8(in.GetLine()) to get a valid Utf from the input for example. Just need to un-error-escape on store. Again, these two different steps will not be necessary in normal apps.

Do you find any utility in this (and not from a GUI programmers stand-point, but a generic library's stand-point)?