Subject: Re: 16 bits wchar
Posted by cbpporter on Wed, 03 Oct 2007 04:16:38 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 01 October 2007 14:28
I guess fixing Utf8 routines to provide UTF16 surrogate support (for now) is a good idea.

Great! On the side note though, I am extending Utf8 methods to handle 4 byte long encodings, not UTF-16 surrogate pairs (which are illegal in UTF-8).

Quote:
Also, I do not think that any string manipulation routine everywhere ever should be aware about UTF-8 or UTF-16 encoding. It is much practical to covert to WString, process and (eventually) convert it back. I think that in the long run, it might be even faster.

That could be an acceptable compromise. But a few processing functions couldn't hurt when you really want to process that string in place.

Quote:
Anyway, seriously, I believe that the ultimate solution is to go with sizeof(wchar) = 4... The only trouble is converting this to UTF16 and back in Win32 everywhere... OTOH, good news is that after the system code is fixed, the transition does not pose too much backwards compatibility problems...

I think you should keep UTF-16 as default for Win32 and UTF-32 as default for Linux. Win32 and .NET both use UTF-16 (with surrogates - Win98 doesn't support surrogates, but the rest do), so I think the future of character encoding for GUI purposes is pretty well defined.

I started working on GetUtf8. I tried to keep everything as close to your style of designing things, but I have two questions.

1. I couldn’t find any function that reads or writes UTF-8 strings (only a single char). The rest of the functions read using pain byte storage. This is OK for storing strings, but when loading them, I need a UTF-8 aware method.

2. Your GetUtf8 method is quite straightforward, but I'm afraid it does not decode values correctly.

Here is a pseudo code of what you do:

```
if(code <= 0x7F)
   compute 1 byte value
else if (code <= 0xDF)
   compute 2 byte value
else if (code <= 0xEF)
   compute 3 byte value
```

else if (...)
    pretty much just read them and return "space"

The issue with this is for the invalid value range 80-C1 which is handled by you second if clause. These values are invalid in UTF-8, but you still decode them using their value and the value of the next character. If this is done for error-escaping, the UTF-8 standard expects to error escape only the current character and start procesing the next one and not to build the error escaped code by using more than the absolute minimum number of code-units (in this case one).
