Subject: More Unicode questions
Posted by cbpporter on Thu, 21 May 2009 11:52:58 GMT
View Forum Message <> Reply to Message

With my never ending quest of screwing around with encodings and Unicode I thought I'd try an experiment and clean up some small parts that deal with such issues, thus both making it easier to eventually move to a full Unicode system and introduce a NOASCII flag (since I can't put off the adoption of Vista and especially 7 forever) and making some part of my <windows.h>-less fork more easy to maintain, since that is the version I use during development.

So I started with:

```
bool FileDelete(const char *filename)
{
#if defined(PLATFORM_WIN32)
	if(IsWinNT())
		return !!UnicodeWin32().DeleteFileW(ToSystemCharsetW(filename));
	else
		return !!DeleteFile(ToSystemCharset(filename));
#elif defined(PLATFORM_POSIX)
	return !unlink(ToSystemCharset(filename));
#else
	#error
#endif//PLATFORM
}
```

as a nice point to adapt. It calls Win API to delete a file, but before it converts the name to the system encoding.

But I'm having some problems understanding what is happening, especially with ToSystemCharset:

```
#ifdef PLATFORM_WIN32
String ToSystemCharset(const String& src)
{
	WString s = src.ToWString();
	int l = s.GetLength() * 5;
	StringBuffer b(l);
	int q = WideCharToMultiByte(CP_ACP, 0, (const WCHAR *)~s, s.GetLength(), b, l, NULL, NULL);
	if(q <= 0)
		return src;
	b.SetCount(q);
	return b;
}
```

So basically in pre Unicode Windows you have two system wide code pages: OEM for consoles and ANSI for GUI. This tells the system which encoding to use with 8-bit strings.

So let's say a have a filename F which need to be passed to DeleteFile. It is in a given encoding, and since on the first line of ToSystemCharset it is converted to a wide string using the default encoding, it must be also in the default encoding or the conversion won't make any sense. On pre Unicode Windows I'm guessing that default encoding is the system wide ANSI code page, and on Unicode Windows it is either Utf-8, or more likely the system wide encoding for non Unicode applications, which plays the same role as on pre Unicode systems. So we have the filename F encoded in encoding C. This is converted to a wide char string, which I'm guessing is Utf16. Then this wide string is converted with WideCharToMultiByte to the ANSI system code page. So basically you're converting F from encoding C to encoding C.

The only situation this makes sense it if the default encoding for strings is changed to something different than the system wide. I'm sorry if a misunderstood this.

And a separate idea: wouldn't it make sense if all 8bit strings were Utf-8 internally? Even on Windows 98 you could still convert them to the system encoding in the few cases where Windows API is called, basically the same idea and approach that is used right now. The only disadvantage would be that some national Latin characters would be 2 bytes long. On the other hand, as helpful as constant length 1 byte chars are, perpetuating the legacy encoding system in U++ internals is not necessary the best idea.

So basically what impact would there be if all read string would be converted to Utf8 using the system code page and stored that way, and converted back to their encoding when calling Win98 API and to Utf16 when calling NT API? I know that this pretty much is happening right now, but with my proposal having a String with different non Utf8 encoding would become impossible. Sure, one could manually convert it to a desired encoding, but the default and what all U++ function would accept would be Utf8 (and Utf16 for wide strings).

I'm just trowing ideas around. The only part that counts is that I need to identify and tweak functions like FileDelete so that they are compilable for either ASCII/Unicode mode or just Unicode mode. FileDelete could become something like:

```
bool FileDelete(const char *filename)
{
#if defined(PLATFORM_WIN32)
#if defined(flagNOASCII)
 return !!DeleteFileW(ToSystemCharsetW(filename));
#else
    if(IsWinNT())
        return !!UnicodeWin32().DeleteFileW(ToSystemCharsetW(filename));
    else
        return !!DeleteFile(ToSystemCharset(filename));
#endif
#elif defined(PLATFORM_POSIX)
    return !unlink(ToSystemCharset(filename));
#else
    #error
#endif//PLATFORM
```

}
I could go through every such function and do the necessary modifications. Also such binaries compiled with NOASCII flag would still run on Win98 where the MS Unicode Compatibility layer is installed (unicows.dll I believe it is called).

Also there are some function that call Win API and do not do the necessary code page transformation. IMO this is a bug.