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.

Subject: Re: More Unicode questions
Posted by mirek on Thu, 21 May 2009 14:46:49 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 21 May 2009 07:52
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.

Exactly. Obvious example is an application in Win98 that is internally Utf-8...

Quote:
And a separate idea: wouldn't it make sense if all 8bit strings were Utf-8 internally?

It would. In fact, I think there is even the recommendation that application should prefer Utf-8 as internal encoding.

Anyway, there are legacy applications that are not utf-8 and that we need to support... And it really is not very complicated to add support for non-Utf8 internal encoding - you need ToSystemCharset[W] no matter what, all you need to do is to have some global default charset variable (SetDefaultCharset) and test
it in ToSystemCharset[W].

Mirek

Subject: Re: More Unicode questions
Posted by cbpporter on Fri, 22 May 2009 10:56:41 GMT
View Forum Message <> Reply to Message

I started investigating this problem because of the Vista Unicode input bug in ANSI applications: other post.

But I can no longer reproduce this bug. It can be one of the following things:
1. It was fixed in U++. The definition of an Unicode/ANSI application is a little bit vague, since you can mix and match W/A version. Probably using CreateWindowW and related versions would

make the bug disappear. Did something get fixed in this part? I can see than CreateWindowA and W are used arbitrarily in U++ sources.
2. They fixed the bug: 936060. It seems to be included in SP1.
3. I'm on a different machine.

I created a test Core with a USEASCII flag, but if this bug is no longer an issues, I don't think it is worth the trouble.

On the other hand, there are still hundreds of places where incorrect ANSI version is used rather then having version selected at runtime. I could fix Core at least so that it always check for correct version.

And I also found a lot of places where Windows API is used instead of equivalent U++ methods, like DeleteFile instead of FileDelete.

Subject: Re: More Unicode questions
Posted by cbpporter on Mon, 25 May 2009 10:49:55 GMT
View Forum Message <> Reply to Message

I started my hunting down of invalid uses of ANSI API on Unicode Windows.

I started with the part that gets the environment variables, namely AppInitEnvironment__. This uses ANSI versions, so even through my environment variables are kept by windows in Unicode, any non Latin characters would be lost in U++ applications. Following the normal procedure, I would test with IsWinNT and duplicate the code, but this time using wchar and W version of the API. Simple enough in theory.

Unfortunately I have bitten of more than I can chew. It seems that even GetEnvironmentStringsW isn't capable of returning Unicode strings. Even worse, result vary a lot depending on system locale, but even a string entered with characters from that locale won't be returned correctly, with '?' characters replacing non Latin ones, but not all of them. I have tried all other Win API functions to retrieve the variables, and the result is the same.

There is only one place where the values are correct: the Windows registry. I've written a test version that in my tests retrieves correctly environment variables with all possible of characters used.

If anybody has time and patience for a short experiment, then please try to create a environment variable with some national characters, or any Unicode character and see if the following code retrieves the correct values:

```
WString GetWinRegStringW(const wchar *value, const wchar *path, HKEY base_key) {
 HKEY key = 0;
 if(RegOpenKeyExW(base_key, path, 0, KEY_READ, &key) != ERROR_SUCCESS)
  return WString::GetVoid();
 dword type, data;
```

```
 if(RegQueryValueExW(key, value, 0, &type, NULL, &data) != ERROR_SUCCESS)
 {
  RegCloseKey(key);
  return WString::GetVoid();
 }
 WStringBuffer raw_data(data);
 if(RegQueryValueExW(key, value, 0, 0, (byte *)~raw_data, &data) != ERROR_SUCCESS)
 {
  RegCloseKey(key);
  return WString::GetVoid();
 }
 if(data > 0 && (type == REG_SZ || type == REG_EXPAND_SZ))
  data -= 2;
 raw_data.SetLength(data / 2);
 RegCloseKey(key);
 return raw_data;
}

void Win32GetEnvVars()
{
#ifndef PLATFORM_WINCE
 //FileOut f("c:\\test.log");
 if (IsWin2K()) {
  wchar *env = GetEnvironmentStringsW();
  for(wchar *ptr = env; *ptr; ptr++)
  {
   const wchar *b = ptr;
   if(*ptr)
    ptr++;
   while(*ptr && *ptr != '=')
    ptr++;
   WString varname(b, ptr);

   if(*ptr)
    ptr++;
   b = ptr;
   while(*ptr)
    ptr++;
   WString value = WString(b, ptr);

   WString newval = GetWinRegStringW(varname,
L"SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment",
HKEY_LOCAL_MACHINE);
   if (!newval.IsVoid())
    value = newval;

   newval = GetWinRegStringW(varname, L"Environment", HKEY_CURRENT_USER);
   if (!newval.IsVoid())
```

```
    value = newval;

   /*WString temp = (varname + "=" + value);
   WString nl = L"\r\n";
   f.PutW(temp, temp.GetCount());
   f.PutW(nl, 2);*/

   coreEnvPtr__().GetAdd(ToUpper(varname).ToString()) = value.ToString();
  }
  FreeEnvironmentStringsW(env);
 }
 else {
  char *env = GetEnvironmentStringsA();
  for(char *ptr = env; *ptr; ptr++)
  {
   const char *b = ptr;
   if(*ptr)
    ptr++;
   while(*ptr && *ptr != '=')
    ptr++;
   String varname(b, ptr);

   if(*ptr)
    ptr++;
   b = ptr;
   while(*ptr)
    ptr++;
   coreEnvPtr__().GetAdd(ToUpper(varname)) = FromSystemCharset(String(b, ptr));
  }
  FreeEnvironmentStringsA(env);
 }
 //f.Close();
#endif
}

void AppInitEnvironment__() {
 Win32GetEnvVars();
 CommonInit();
}
```

The code replaces AppInitEnvironment__ in Core/App.cpp.

Thank you!

Subject: Re: More Unicode questions
Posted by cbpporter on Mon, 25 May 2009 12:49:05 GMT

And more fixes. Tested under XP/Vista. Don't have 95/98.

```
String GetComputerName()
{
#if defined(PLATFORM_WIN32)
 if (IsWinNT()) {
  wchar temp[256];
  *temp = 0;
  dword w = 255;
  ::GetComputerNameW(temp, &w);
  return WString(temp).ToString();
 }
 else
 {
  char temp[256];
  *temp = 0;
  dword w = 255;
  ::GetComputerNameA(temp, &w);
  return FromSystemCharset(temp);
 }
#else
 char temp[256];
 *temp = 0;
 gethostname(temp, sizeof(temp));
 return FromSystemCharset(temp);
#endif
}

String GetUserName()
{
#if defined(PLATFORM_WIN32)
 if (IsWinNT()) {
  wchar temp[256];
  *temp = 0;
  dword w = 255;
  ::GetUserNameW(temp, &w);
  return WString(temp).ToString();
 }
 else {
  char temp[256];
  *temp = 0;
  dword w = 255;
  ::GetUserNameA(temp, &w);
  return FromSystemCharset(temp);
 }
#else
```

```
 return Nvl(GetEnv("USER"), "boot");
#endif
}
```

Subject: Re: More Unicode questions
Posted by cbpporter on Tue, 26 May 2009 12:29:50 GMT
View Forum Message <> Reply to Message

And for GetEnv:

```
String GetEnv(const char *id)
{
#if defined(PLATFORM_WIN32)
 if (IsWinNT()) {
  WStringBuffer temp(32767);
  WString varname(id);
  int len = GetEnvironmentVariableW(varname, temp, 32766);
  temp.SetLength(len);
  WString value = temp;

  WString newval = GetWinRegStringW(varname,
L"SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment",
HKEY_LOCAL_MACHINE);
  if (!newval.IsVoid())
   value = newval;

  newval = GetWinRegStringW(varname, L"Environment", HKEY_CURRENT_USER);
  if (!newval.IsVoid())
   value = newval;

  return value.ToString();
 }
 else
  return FromSystemCharset(getenv(id));
#else
 return FromSystemCharset(getenv(id));
#endif
}
```

Subject: Re: More Unicode questions
Posted by cbpporter on Thu, 28 May 2009 12:06:10 GMT
View Forum Message <> Reply to Message

There was a bug in previous GetEnv post regarding not setting the buffer to smallest size which caused problems in other paces. I have edited it and hopefully it is correct now. I'll review everything again to make sure that it works.
