Subject: Re: 16 bits wchar
Posted by cbpporter on Tue, 25 Sep 2007 20:03:09 GMT
View Forum Message <> Reply to Message

I was quite unhappy when I found out that U++ is not Unicode standard compliant with it's "UTF-16" (what it implements is actually UCS-2). There are o lot of programs with poor Unicode support, which is partyially because STL doesn't support full Unicode either.

In theory it would be quite unforgivable for an application to handle just a subset of the standard. But how does the situation look in theory?

To answer this question I did a number of relatively thorough tests which took me about two hours. I used my computer at work, which has a Windows XP SP2 operating system. The first part was to determine if the OS supports surrogate pairs. After some testing (and research) I found that surrogate pairs can be enabled easily and are enabled by default. Windows has no problems theoretically to use the kind of characters (but individual pieces of software can). Next I found a font which displays about 20000 characters with codes about 0xFFFF, I installed them, and surprise surprise, it worked.

Next I tested a couple of applications. At first I wanted to give exact results, but I found it boring to write them and concluded that you will find it boring to read them. In short, Notepad and WordPad both display correctly and identify two code-units as one code point. Opera doesn’t identify code points correctly in some files and it can no do copy operations (it will truncate only to the lower 16 bits). Internet Explorer works correctly, but it couldn't use the correct registry entries to display the characters, so it used a little black rectangle. And the viewer from Total Commander is really ill equipped for these kinds of tasks.

Next I would like to test U++, but I get strange results when trying to find the length of a string when using only normal characters (with codes below 0xFFFF).

I took one of the examples and slightly modified it:

```
#include <Core/Core.h>

using namespace Upp;

CONSOLE_APP_MAIN
{
 SetDefaultCharset(CHARSET_UTF8);

 WString x = "";
 for(int i = 280; i < 300; i++)
  x.Cat(i);
 DUMP(x);
 DUMP(x.GetLength());
 DUMP(x.GetCount());

 String y = x.ToString();
 DUMP(y);
 DUMP(y.GetLength());
 DUMP(y.GetCount());

 y.Cat(" (appended)");
 x = y.ToWString();

 DUMP(x);
 DUMP(x.GetLength());
 DUMP(x.GetCount());
}
```

I got these results:

```
x = 


x.GetLength() = 20
x.GetCount() = 20
y = 


y.GetLength() = 40
y.GetCount() = 40
x = 


x.GetLength() = 31
x.GetCount() = 31
```

Except the fact that the cars are mangled, the lengths doesn't seem to be ok. I may have understood incorrectly, but AFAIK GetLength should return the length in code units and GgtCount the number of real characters, so code points.

I also started researching the exact encoding methods of UTF and I will add full Unicode support to strings. It will be for personal use, but if anybody is interested I will post my results. Right now I'm trying to find and efficient way to index multichar strings. I think I will have to use iterators instead.