Subject: Re: 16 bits wchar
Posted by mirek on Wed, 03 Oct 2007 10:10:35 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 03 October 2007 04:36luzr wrote on Wed, 03 October 2007 10:26Error escaping in Stream:

The error escaping in GetUtf8 is impossible, as it returns only single int - you do not know you have to escape until you read more than single character from the input - and then you need more than one wchar to be returned...
It depends on what that int represents and what kind of error escaping is used. For Utf-8, there are only a small number of characters that are invalid and they could be escaped to non-character code-points or even to a small region of the Private Use Area (for example FFF00-FFFFF). The private user area has approximatively 130000 reserved code points which are guaranteed to not appear in public Unicode data (they are reserved for private processing only, not data interchange).

Ah, but that is not the problem - AFAIK.

The trouble is e.g. invalid 6 bytes sequence, which you detect at byte 6. In this case, you cannot reasonable return anything escaped from Stream::GetUtf8. You would need more than 32-bit value for any reasonable output.

BTW, private area is exactly what "real" Utf8 functions use, just the range is 0xEE00 - 0xEEFF (did not wanted to spoil the beginning of range and 0xEExx nicely resonates with "Error Escape"

However, please check the fixed version Stream::GetUtf8():

```
int Stream::GetUtf8()
{
 int code = Get();
 if(code <= 0) {
  LoadError();
  return -1;
 }
 if(code < 0x80)
  return code;
 else
 if(code < 0xC0)
  return -1;
 else
 if(code < 0xE0) {
  if(IsEof()) {
   LoadError();
   return -1;
  }
  return ((code - 0xC0) << 6) + Get() - 0x80;
```

```
}
else
if(code < 0xF0) {
 int c0 = Get();
 int c1 = Get();
 if(c1 < 0) {
  LoadError();
  return -1;
 }
 return ((code - 0xE0) << 12) + ((c0 - 0x80) << 6) + c1 - 0x80;
}
else
if(code < 0xF8) {
 int c0 = Get();
 int c1 = Get();
 int c2 = Get();
 if(c2 < 0) {
  LoadError();
  return -1;
 }
 return ((code - 0xf0) << 18) + ((c0 - 0x80) << 12) + ((c1 - 0x80) << 6) + c2 - 0x80;
}
else
if(code < 0xFC) {
 int c0 = Get();
 int c1 = Get();
 int c2 = Get();
 int c3 = Get();
 if(c3 < 0) {
  LoadError();
  return -1;
 }
 return ((code - 0xF8) << 24) + ((c0 - 0x80) << 18) + ((c1 - 0x80) << 12) +
        ((c2 - 0x80) << 6) + c3 - 0x80;
}
else
if(code < 0xFE) {
 int c0 = Get();
 int c1 = Get();
 int c2 = Get();
 int c3 = Get();
 int c4 = Get();
 if(c4 < 0) {
  LoadError();
  return -1;
 }
 return ((code - 0xFC) << 30) + ((c0 - 0x80) << 24) + ((c1 - 0x80) << 18) +
        ((c2 - 0x80) << 12) + ((c3 - 0x80) << 6) + c4 - 0x80;
```

```
 }
 else {
  LoadError();
  return -1;
 }
}
```

BTW, thinking further about UTF-8 -> UTF-16 surrogate conversion, I am afraid that it in fact can cause some problems in the code.

The primary motivation for "Error Escape" is that when file that is not representable by UCS-2 wchars is loaded into the editor (e.g. IDE) or if it simply has UTF-8 errors, there are two requirements:

- Parts of file with correct and representable UTF-8 encoding must be editable

- Invalid parts must not be damaged by loading/saving.

I am afraid that with real surrogate pairs in editor, editor logic can go bad, it really expects that single wchar represents one code point. There would be visual artifacts, with Win32 interpretting surrogate pairs correctly (while U++ considering them single characters).

What a nice bunch of problems to solve And we have not even started to consider REAL problems

Mirek