Discussion of how to add UTF-16 support to a library that internally uses UTF-8

devblogs.microsoft.com/oldnewthing/20170714-00

July 14, 2017

Raymond Chen

A customer was incorporating an external library that internally manages file names in UTF-8. They wanted to add UTF-16 support[1] and to avoid having to make a lot of changes to the library to change its internal data format, they figured it'd be less risky to keep the internal format as UTF-8 and convert the file names from UTF-16 to UTF-8 as they enter the library, and convert them from UTF-8 back to UTF-16 whenever the library needs to call out to Windows (for example, by passing the file name to the `CreateFile` function).

The customer wanted to know if there were any pitfalls to this approach. In particular, is it guaranteed that converting a UTF-16 string to UTF-8 and then converting back to UTF-16 will result in a string that is byte-for-byte identical to the original?

Shawn Steele replied that the conversion is reversible, provided that the original UTF-16 string is valid. He cautioned that sometimes people are under the false impression that a UTF-8-encoded string or a UTF-16-encoded string can contain arbitrary binary data. As a result, they end up passing things like unmatched high and low surrogates (for UTF-16) or improper continuation bytes (for UTF-8). There might also be incorrect substring or string concatenation algorithms which expect that a string can be chopped at any point and produce a meaningful result.

He also pointed out that many characters have multiple encodings. For example, "Ä" can be encoded as the single code point U+00C4 (LATIN CAPITAL A WITH DIAERESIS) or as the sequence of code points U+0041 (LATIN CAPITAL A) followed by U+0308 (COMBINING DIAERESIS).[2]

The customer thanked Shawn for his advice. They had already encountered the second problem (known as Normalization), but since the Windows file system does not perform normalization, they figured their program shouldn't do it either. They were a bit concerned about the issue with substrings and string concatenation and were wondering if this was a case where `_mbscat_s` would be used instead of `strcat_s`.

This led to an extended discussion about surrogate pairs, zero-width-joiners, extended grapheme clusters, and stop ascribing meaning to Unicode code points.

I stepped in and tried to return to the customer's question. All of these issues with substrings and concatenation and extended grapheme clusters are issues for the library itself, not for the UTF-16 wrapper the customer is building. If there are any problems in the library, they can raise them with the maintainers of the library.

The customer wanted a UTF-16 entry point to the library which forwards to the existing UTF-8 entry point. In that case, they can call `MultiByteToWideChar` with the `MB_ERR_INVALID_CHARS` flag and return an appropriate failure if the string is not well-formed. If the string successfully converts from UTF-16 to UTF-8, then it will also successfully convert back from UTF-8 to UTF-16 with no loss of fidelity.[3]

[1] In the context of Windows, Unicode strings are encoded in UTF-16LE if not explicitly called out otherwise.

[2] I find it somewhat quaint that names of Unicode code points are written in all-caps.

[3] The customer thanked both Shawn and me for our assistance, even though my contribution was basically to take the long discussion and focus the answer to the customer's actual problem. I confessed that I didn't add any information; I merely deleted the distractions. Shawn replied, "Yea, but you're better at saying things shorter than I can."

Raymond Chen

Follow