

Re: Unicode Support

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2005-04/msg00683.html>

- *From:* websnarf@xxxxxxxxxx
 - *Date:* 19 Apr 2005 20:59:22 -0700
-

Chewy...@xxxxxxxxxxxxxxxxxxx wrote:

> websnarf@xxxxxxxxxx wrote:
>> Chewy509@xxxxxxxxxxxxxxxxxxx wrote:
>>> Something that has been bugging me, since I started on my
>>> own compiler/assembler is unicode support. Not the API's
>>> or libraries, or how unicode works (it's quite simple
>>> once you get your head around it),
>>
>> Are you sure about that?
>
> Having read the spec a few times, it doesn't seem that hard?
> (combinations and normalisations do seem to be the most
> complex issues though).

Yeah see ... you kind of HAVE to implement that stuff (actually you probably just need normalization). Otherwise how are you going to know if two Unicode strings are the same? (As I said, just the ordinary action of symbol matching boils down to writing a string comparison function for Unicode strings -- did you know that you can write e with an accent grave on it in two different ways?)

> All encoding methods, whether that be UTF-8, UTF-16 or UTF-32
> handle the same number of encodings, (1024K), just the way
> that they are held in memory/disk is a little different.

Actually its slightly more than 1024K. But currently only about 100K of them are assigned.

> [...] I wouldn't call UTF-8 "the more
> international approach", but rather "the lets add backwards
> compatibility for the those that still believe ASCII is the
> only real encoding format approach".

Right -- the point is that even internationally, ASCII is used somewhat. UTF-16 is basically telling everyone "ok we all got to start from scratch all over again!" and was pushed, mainly by an american consortium. UTF-16 is a generalization of the original UCS-2, which only encoded code points up to 0xffff (the BMP? Who cares, its obsolete) and which was fine for Microsoft, et al's first attempt at

Re: Unicode Support

internationalization. This worked great for americans, and europeans, but kind of left the asians with a bad taste in their mouth. UTF-8 is not tied to a stunted encoding -- it was right from the very beginning.

>> - UTF-16 requires a "BOM" at the beginning of any string to
>> distinguish endianness, which means that if you split a string
>> into two pieces, then transmit them, you end up gaining an
>> additional "BOM" character for the second piece.
>
> A BOM is not necessary, when the encoding is known or dictated
> by implementation. Since assemblers generally operate on the
> target CPU (I know there are exceptions, such as cross-compilers
> and cross-assemblers), it can be fair to say that the format can
> be dictated by the assembler, whether that be UTF-16BE or
> UTF-16LE (LE for x86).

Ok, first of all, cross assemblers exist just as much as cross compilers. Second of all, this is about what your *text editor* supports, not your OS. The text editors will likely output a BOM character, and will likely support *both* endians.

>> - UTF-8 directly supports ASCII encoding as a sub-mode of
>> its encoding. I.e., normal ASCII encoded text is *already*
>> UTF-8 compatible. Certain ASCII functions like changing
>> english text case, or searching for ASCII characters can
>> be done directly on UTF-8 data. So '\0' termination,
>> tabs, or things like CR and LF don't have strange
>> embodiments or representations, even when viewed with
>> ASCII tools. UTF-8 encodings are also easy to learn to
>> recognize on sight, even with ASCII tools.

>
> Granted.

>
>> - UTF-8 can be "resynched" even if a transfer channel is
>> corrupted, or if you start from the middle of the string
>> after only a very short (I think at most 5) character
>> scan. Compare this to UTF-16, where if you suddenly
>> become offset by one character due to some flaw/error,
>> you will have no idea that your data is all corrupted
>> for an unbounded length of time.

>
> Are you sure?

Yeah, that's how it was designed; let me check. If the top bit is 0, then the byte is a character in the range 0x00-0x7f (i.e., its ASCII). If the top two bits are 11, then the byte is the *start* of a multibyte encoding of a UTF-8 character. If the top two bits are 10 then it is a continuation character at there are at most *2* continuation characters following it (for the legal Unicode range.) So, in fact, its at most 3 characters for a resynch of a unidirectional channel, or a at most two characters in each direction in additional to the current character

Re: Unicode Support

(that's 5 total) for a bidirectional channel.

- > UTF-16 is easier than UTF-8 (and only requires 1 backstep
- > at most), and this doesn't apply to UTF-32 at all. The
- > way surrogate pairs work in UTF-16, it's just a quick
- > test to see if a bit is 1 or 0 to determine first or
- > second character.

No, no, no. A channel is usually encoded as bytes. If you get offset by a single byte, then you will just read different data as UTF-16. Since most of the encodings of UTF-16 are in fact completely legal and since it does not have any kind of self-integrity encoding features like UTF-8 does, you will likely have no inkling that an error has occurred until far far later that the original error actually occurred.

You cannot assume that "errors" will just nicely cause you to skip an even number of bytes all the time.

- >> Not surprisingly, Microsoft supports UTF-16 pervasively.
- >
- > Surprising MS adopted Unicode *before* Unicode 1.0 was
- > finalised. (Which is what lead to some of the quirks of
- > the MS Unicode implementation).

Yes, as I said, Microsoft, and a number of other american companies really got on the bandwagon of Unicode waaaaaay too soon. In fact I would consider the *current* standard, the equivalent of a Beta release (whereas each earlier release was an Alpha, or prototype.) ISO 10646 was clearly the superior evolution path.

> <snip>

>

- >>> PS. If your assembler already supports UTF-16 based
- >>> source code, I would be deeply interested in hearing
- >>> about some of the challenges in implementing unicode
- >>> support. In particular, did you limit numbers to the
- >>> western 0..9 figures, or did you allow other numbers
- >>> to be included, eg arabic, many of the asian sets, etc.
- >>> Did you limit to valid range of characters to the BMP
- >>> (the first 64K characters only), or did you allow for
- >>> the full range of characters (1024K characters) for
- >>> labels. How did you handle compatible encodings, and
- >>> combining characters? What about UTF-8 vs UTF-16 vs
- >>> UTF-32?
- >>
- >> UTF-32 is mostly useful from a programming internal
- >> format. I.e., I don't think supporting it for source
- >> code encodings is worth while at all (since its so
- >> inefficient.) But for data encodings, I would
- >> recommend supporting all three of them (since
- >> programmers may want to use any of the modes in their

Re: Unicode Support

>> programs).
>
> I agree. UTF-32 for source would be a waste. Internally, it
> would help somewhat, but I don't think the overhead for the
> <1% of cases would be worth it. The Unicode standard does
> allow for implementations only to handle the BMP (U+0000
> .. U+FFFF), and still be conformant. Maybe that's an
> option?

No its not. That's the *mistake* Microsoft and Sun started with. Its just an insult to the asians to ignore the encodings beyond U+FFFD (U+FFFE and U+FFFF are illegal.) Remember, that Unicode people are writing things in a way to try to try to cover up their mistakes from the past. Just do it the right way, and let Microsoft deal with the legacy problems they've made for themselves. At least UTF-16 is backward compatible with the old Unicode standard.

>>> PPS. I know the DOS API doesn't support unicode
>>> strings, but just used it for the example.
>>
>> Well that's actually a kind of non-trivial point. If you
>> support Unicode as datatype (no reason why you couldn't)
>> there is the question of what APIs do you intend to pass
>> this data around in?
>>
>>> PPPS. The full Unicode 4.1 spec can be downloaded as
>>> PDF's from www.unicode.org.
>>
>> Yeah, so is version 4.0, 3.1, 3.0, ... etc. Taking a step
>> back, one of the real problems with Unicode is that its
>> rate of evolution is unusually high for such an important
>> and universal standard.
>
> However the standard does make references for further
> feature compatibility. Eg any implementation that is Unicode
> 3.0 conformant, will also be Unicode 4.x conformant. While
> it is an issue, I don't believe there is enough risk evolved
> to warrant too much time on it.

There are a number of proposed character sets still pending for approval for the next update. (One includes an interesting alphabet designed by some University of Toronto researchers to help retarded people.) The standard does not say anything about whether or not any future encodings will include other redundant encodings that further complicates normalization.

--
Paul Hsieh
<http://www.pobox.com/~qed/>
<http://bstring.sf.net/>

- *Follow-Ups:*
 - ◆ **Re: Unicode Support**
 - ◇ *From:* Chewy509

- *References:*
 - ◆ **Unicode Support**
 - ◇ *From:* Chewy509
 - ◆ **Re: Unicode Support**
 - ◇ *From:* websnarf
 - ◆ **Re: Unicode Support**
 - ◇ *From:* Chewy509

- Prev by Date: **Re: Unicode Support**
- Next by Date: **Re: Unicode Support**
- Previous by thread: **Re: Unicode Support**
- Next by thread: **Re: Unicode Support**
- Index(es):
 - ◆ **Date**
 - ◆ **Thread**