POSIX Liaison Issue

The Austin Group has recently been considering the question posed in http://austingroupbugs.net/view.php?id=663: Specification of str[n]casecmp is ambiguous

The submitter of this issue states

> The description [of strcasecmp] includes the text:
>
> "When the LC_CTYPE category of the current locale is from the POSIX locale, strcasecmp() and strncasecmp() shall behave as if the strings had been converted to lowercase and then a byte comparison performed. Otherwise, the results are unspecified."
>
> As far as I can tell, the phrase "converted to lowercase" is not precisely specified anywhere. This is not a serious problem in the event that the POSIX locale only contains the required characters, but per XBD 7.2, implementations are permitted to have other characters available in the POSIX locale ("The tables in Locale Definition describe the characteristics and behavior of the POSIX locale for data consisting entirely of characters from the portable character set and the control character set. For other characters, the behavior is unspecified.")
>
> If additional characters outside the portable character set exist in the POSIX locale, does "converted to lowercase" mean as if by tolower on a byte-by-byte basis, or as if by towlower on a character-by-character basis? Or does the "unspecified" clause in XBD 7.2 leave this choice up to the implementation?
>
> A more worrisome issue is, in the case that an implementation has multibyte characters in the POSIX locale, the symmetry of strncasecmp is broken. The n argument is specified as the maximum number of bytes to compare from s1, not from s2, so from my reading of the text as written, strncasecmp may read more than n bytes from s2, and strncasecmp(a,b,n)>0 is not necessarily equivalent to strncasecmp(b,a,n)<0. In particular, consider the example (assuming UTF-8):
>
> strncasecmp("\u00df", "\u1e9e", 2)

When considering this, it should be noted that the POSIX locale and the C locale are intended to be aligned. The following comment was submitted to help us in considering this issue:

> POSIX is already clear that the contract of isupper() defers to the C standard [line 38829], and the C standard is already explicit [C99 7.4.1.11] that in the C locale, isupper() may return true only for the characters which are letters according to C99 5.2.1. However, in re-reading 5.2.1, I see the following:
>
> There is a requirement that there are exactly 26 uppercase letters in the basic execution character set. Likewise, there is a requirement that there are exactly 10 decimal digits in the basic execution character set, with consecutive encodings. There is also a strong statement that:
>
> "A letter is an uppercase letter or a lowercase letter as defined above: in this International Standard the term does not include other characters that are letters in other alphabets."

> We were originally interpreting this to mean that the C locale provides exactly the basic execution character set, with exactly 26 uppercase letters, and cannot provide extension letters. However, on re-reading, that statement in the C standard sounds to me like a letter is either one of the 26 uppercase letters in the basic character set, OR a character in the extended character set that is a letter, and merely that if the extended character set is smaller than the set of Unicode characters, then the term 'letter' in the standard would not include those Unicode letters not in the implementation's extended character set. But if we read it that way, where the extended character set can provide additional letters, then that also means that the extended character set can provide additional decimal digits.
>
> Yet, in the POSIX standard, we required that there are exactly 10 characters that can belong to the 'digit' classification of the C locale [line 4015]. The approach in Note: 0001501 was to extend the exactness already present on digits to also apply to letters, but maybe the right approach is to instead go the other direction and not only allow extension character set members as additions to the set of letters, but also allow extension character set members as additions to the set of decimal digits. This would mean that you can no longer user c-'0' to find the numeric value corresponding to a character where isdigit(c) returns non-zero.
>
> Now, you DO have some constraints - my understanding is that Unicode includes some characters which are considered letters, but which are neither uppercase nor lowercase. However, while non-C locales are permitted to have a character where isupper(c) and islower(c) both return false, but isalpha(c) returns true, this is forbidden on the C locale.

The liaison questions are:

- Is the Austin Group Correct in its interpretation of the C standard as described above?
- Does WG14 believe that the C locale can contain multi-byte characters? If so, what is the rationale for this?
- Does the committee agree with the interpretation response proposed at http://austingroupbugs.net/view.php?id=663#c1501?”

For reference, here is the text of our currently proposed resolution:

> Interpretation response
> ----------
> The standard clearly states that the POSIX locale is a synonym for the C locale with regards to LC_CTYPE, and defers to the C standard for its definitions of character categories. The C standard is clear that only 26 uppercase and 26 lowercase letters exist in the C locale, all of which have single-byte encodings.
>
> Rationale:
> ----------
> While the POSIX locale may include multibyte characters, it must still honor the C locale rules. Since the C locale has only 26 characters that can be altered by tolower(), all of which are single bytes, it does not matter whether strcasecmp() and strncasecmp() use byte or character normalization, and the length limit of strncasecmp() is specified in bytes.
>
> We are also seeking proposals for the standardization of a "POSIX.UTF-8" locale for Issue 8.

Notes to the Editor (not part of this interpretation):

Make the following changes:

On page 136 line 3849 [7.2 POSIX locale], move the following sentences:

The tables in Section 7.3 describe the characteristics and behavior of the POSIX locale for data consisting entirely of characters from the portable character set and the control character set. For other characters, the behavior is unspecified.

to page 151 line 4531 [7.3.2.6 LC_COLLATE Category in the POSIX Locale], with a wording change:

The definition below describes the behavior of the POSIX locale for data consisting entirely of characters from the portable character set and the control character set. For other characters, the behavior is unspecified.

On page 139 line 3996 [7.3.1 LC_CTYPE upper], change:

In the POSIX locale, the 26 uppercase letters shall be included:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

to:

In the POSIX locale, only:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
shall be included.

On page 139 line 4003 [7.3.1 LC_CTYPE lower], change:

In the POSIX locale, the 26 lowercase letters shall be included:
a b c d e f g h i j k l m n o p q r s t u v w x y z

to:

In the POSIX locale, only:
a b c d e f g h i j k l m n o p q r s t u v w x y z
shall be included.

On page 139 line 4009 [7.3.1 LC_CTYPE alpha], change:

In the POSIX locale, all characters in the classes upper and lower shall be included.

to:

In the POSIX locale, only characters in the classes upper and lower shall be included.

On page 141 line 4091 [7.3.1 LC_CTYPE toupper], change:

In the POSIX locale, at a minimum, the 26 lowercase characters:

to:

In the POSIX locale, the 26 lowercase characters:

On page 142 line 4105 [7.3.1 LC_CTYPE tolower], change:

In the POSIX locale, at a minimum, the 26 uppercase characters:

to:

In the POSIX locale, the 26 uppercase characters:

On page 143 line 4145 [7.3.1.1 LC_CTYPE Category in the POSIX Locale], change:

```
# The following is the POSIX locale LC_CTYPE.
# "alpha" is by default "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by default "alnum", "punct", and the <space>
# "graph" is by default "alnum" and "punct"
```

to:

```
# The following is the minimum POSIX locale LC_CTYPE; implementations may
# add additional characters to the "cntrl" and "punct" classifications.
# "alpha" is by definition "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by definition "alnum", "punct", and the <space>
# "graph" is by definition "alnum" and "punct"
```

At page 151 line 4534 [7.3.2.6 LC_COLLATE Category in the POSIX Locale], change:

```
# This is the POSIX locale definition for the LC_COLLATE category.
# The order is the same as in the ASCII codeset.
```

to:

```
# This is the minimum input for the POSIX locale definition for the
# LC_COLLATE category. Characters in this list are in the same order
# as in the ASCII codeset.
```

NOTE: additional changes are currently being worked on, but the core of the response is as above.

Thank you all for your consideration of this matter.