            i;unicode-casemap - Simple Unicode Collation Algorithm

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   This document describes "i;unicode-casemap", a simple case-
   insensitive collation for Unicode strings. It provides equality,
   substring, and ordering operations.

1. Introduction

   The "i;ascii-casemap" collation described in [COMPARATOR] is quite
   simple to implement and provides case-independent comparisons for the
   26 Latin alphabetics. It is specified as the default and/or baseline
   comparator in some application protocols, e.g., [IMAP-SORT].

   However, the "i;ascii-casemap" collation does not produce
   satisfactory results with non-ASCII characters. It is possible, with
   a modest extension, to provide a more sophisticated collation with
   greater multilingual applicability than "i;ascii-casemap". This
   extension provides case-independent comparisons for a much greater
   number of characters. It also collates characters with diacriticals
   with the non-diacritical character forms.

   This collation, "i;unicode-casemap", is intended to be an alternative
   to, and preferred over, "i;ascii-casemap". It does not replace the
   "i;basic" collation described in [BASIC].

2. Unicode Casemap Collation Description

   The "i;unicode-casemap" collation is a simple collation which is
   case-insensitive in its treatment of characters. It provides
   equality, substring, and ordering operations. The validity test
   operation returns "valid" for any input.

This collation allows strings in arbitrary (and mixed) character
sets, as long as the character set for each string is identified and
it is possible to convert the string to Unicode. Strings which have
an unidentified character set and/or cannot be converted to Unicode
are not rejected, but are treated as binary.

Each input string is prepared by converting it to a "titlecased
canonicalized UTF-8" string according to the following steps, using
UnicodeData.txt ([UNICODE-DATA]):

   (1) A Unicode codepoint is obtained from the input string.

      (a) If the input string is in a known charset that can be
          converted to Unicode, a sequence in the string's charset
          is read and checked for validity according to the rules of
          that charset. If the sequence is valid, it is converted
          to a Unicode codepoint. Note that for input strings in
          UTF-8, the UTF-8 sequence must be valid according to the
          rules of [UTF-8]; e.g., overlong UTF-8 sequences are
          invalid.

      (b) If the input string is in an unknown charset, or an
          invalid sequence occurs in step (1)(a), conversion ceases.
          No further preparation is performed, and any partial
          preparation results are discarded. The original string is
          used unchanged with the i;octet comparator.

   (2) The following steps, using UnicodeData.txt ([UNICODE-DATA]),
       are performed on the resulting codepoint from step (1)(a).

      (a) If the codepoint has a titlecase property in
          UnicodeData.txt (this is normally the same as the
          uppercase property), the codepoint is converted to the
          codepoints in the titlecase property.

      (b) If the resulting codepoint from (2)(a) has a decomposition
          property of any type in UnicodeData.txt, the codepoint is
          converted to the codepoints in the decomposition property.
          This step is recursively applied to each of the resulting
          codepoints until no more decomposition is possible
          (effectively Normalization Form KD).

      Example: codepoint U+01C4 (LATIN CAPITAL LETTER DZ WITH CARON)
      has a titlecase property of U+01C5 (LATIN CAPITAL LETTER D
      WITH SMALL LETTER Z WITH CARON). Codepoint U+01C5 has a
      decomposition property of U+0044 (LATIN CAPITAL LETTER D)
      U+017E (LATIN SMALL LETTER Z WITH CARON). U+017E has a
      decomposition property of U+007A (LATIN SMALL LETTER Z) U+030c

(COMBINING CARON). Neither U+0044, U+007A, nor U+030C have
any decomposition properties. Therefore, U+01C4 is converted
to U+0044 U+007A U+030C by this step.

(3) The resulting codepoint(s) from step (2) is/are appended, in
UTF-8 format, to the "titlecased canonicalized UTF-8" string.

(4) Repeat from step (1) until there is no more data in the input
string.

Following the above preparation process on each string, the equality,
ordering, and substring operations are as for i;octet.

It is permitted to use an alternative implementation of the above
preparation process if it produces the same results. For example, it
may be more convenient for an implementation to convert all input
strings to a sequence of UTF-16 or UTF-32 values prior to performing
any of the step (2) actions. Similarly, if all input strings are (or
are convertible to) Unicode, it may be possible to use UTF-32 as an
alternative to UTF-8 in step (3).

Note: UTF-16 is unsuitable as an alternative to UTF-8 in step (3),
because UTF-16 surrogates will cause i;octet to collate codepoints
U+E0000 through U+FFFF after non-BMP codepoints.

This collation is not locale sensitive. Consequently, care should be
taken when using OS-supplied functions to implement this collation.
Functions such as strcasecmp and toupper are sometimes locale
sensitive and may inconsistently casemap letters.

The i;unicode-casemap collation is well suited to use with many
Internet protocols and computer languages. Use with natural language
is often inappropriate; even though the collation apparently supports
languages such as Swahili and English, in real-world use it tends to
mis-sort a number of types of string:

o people and place names containing scripts that are not collated
   according to "alphabetical order".
o words with characters that have diacriticals. However,
   i;unicode-casemap generally does a better job than i;ascii-casemap
   for most (but not all) languages. For example, German umlaut
   letters will sort correctly, but some Scandinavian letters will
   not.
o names such as "Lloyd" (which in Welsh sorts after "Lyon", unlike
   in English),
o strings containing other non-letter symbols; e.g., euro and pound
   sterling symbols, quotation marks other than '"', dashes/hyphens,
   etc.

3. Unicode Casemap Collation Registration

```
<?xml version='1.0'?>
<!DOCTYPE collation SYSTEM 'collationreg.dtd'>
<collation rfc="5051" scope="global" intendedUse="common">
<identifier>i;unicode-casemap</identifier>
<title>Unicode Casemap</title>
<operations>equality order substring</operations>
<specification>RFC 5051</specification>
<owner>IETF</owner>
<submitter>mrc@cac.washington.edu</submitter>
</collation>
```

4. Security Considerations

   The security considerations for [UTF-8], [STRINGPREP], and [UNICODE-
   SECURITY] apply and are normative to this specification.

   The results from this comparator will vary depending upon the
   implementation for several reasons. Implementations MUST consider
   whether these possibilities are a problem for their use case:

   1) New characters added in Unicode may have decomposition or
      titlecase properties that will not be known to an implementation
      based upon an older revision of Unicode. This impacts step (2).

   2) Step (2)(b) defines a subset of Normalization Form KD (NFKD) that
      does not require normalization of out-of-order diacriticals.
      However, an implementation MAY use an NFKD library routine that
      does such normalization. This impacts step (2)(b) and possibly
      also step (1)(a), and is an issue only with ill-formed UTF-8
      input.

   3) The set of charsets handled in step (1)(a) is open-ended. UTF-8
      (and, by extension, US-ASCII) are the only mandatory-to-implement
      charsets. This impacts step (1)(a).

      Implementations SHOULD, as far as feasible, support all the
      charsets they are likely to encounter in the input data, in order
      to avoid poor collation caused by the fall through to the (1)(b)
      rule.

   4) Other charsets may have revisions which add new characters that
      are not known to an implementation based upon an older revision.
      This impacts step (1)(a) and possibly also step (1)(b).

An attacker may create input that is ill-formed or in an unknown
charset, with the intention of impacting the results of this
comparator or exploiting other parts of the system which process this
input in different ways. Note, however, that even well-formed data
in a known charset can impact the result of this comparator in
unexpected ways. For example, an attacker can substitute U+0041
(LATIN CAPITAL LETTER A) with U+0391 (GREEK CAPITAL LETTER ALPHA) or
U+0410 (CYRILLIC CAPITAL LETTER A) in the intention of causing a
non-match of strings which visually appear the same and/or causing
the string to appear elsewhere in a sort.

5. IANA Considerations

   The i;unicode-casemap collation defined in section 2 has been added
   to the registry of collations defined in [COMPARATOR].

6. Normative References

   [COMPARATOR]   Newman, C., Duerst, M., and A. Gulbrandsen,
                         "Internet Application Protocol Collation
                         Registry", RFC 4790, February 2007.

   [STRINGPREP]   Hoffman, P. and M. Blanchet, "Preparation of
                         Internationalized Strings ("stringprep")", RFC
                         3454, December 2002.

   [UTF-8]   Yergeau, F., "UTF-8, a transformation format of
                         ISO 10646", STD 63, RFC 3629, November 2003.

   [UNICODE-DATA]   <http://www.unicode.org/Public/UNIDATA/
                         UnicodeData.txt>

                         Although the UnicodeData.txt file referenced
                         here is part of the Unicode standard, it is
                         subject to change as new characters are added
                         to Unicode and errors are corrected in Unicode
                         revisions. As a result, it may be less stable
                         than might otherwise be implied by the
                         standards status of this specification.

   [UNICODE-SECURITY]   Davis, M. and M. Suignard, "Unicode Security
                         Considerations", February 2006,
                         <http://www.unicode.org/reports/tr36/>.

7. Informative References

   [BASIC]   Newman, C., Duerst, M., and A. Gulbrandsen,
                        "i;basic - the Unicode Collation Algorithm",
                        Work in Progress, March 2007.

   [IMAP-SORT]   Crispin, M. and K. Murchison, "Internet Message
                        Access Protocol - SORT and THREAD Extensions",
                        Work in Progress, September 2007.

Author's Address

   Mark R. Crispin
   Networks and Distributed Computing
   University of Washington
   4545 15th Avenue NE
   Seattle, WA 98105-4527

   Phone: +1 (206) 543-5762
   EMail: MRC@CAC.Washington.EDU

Full Copyright Statement

Intellectual Property