NAME

Unicode::Collate - Unicode Collation Algorithm

SYNOPSIS

```
  use Unicode::Collate;

  #construct
  $Collator = Unicode::Collate->new(%tailoring);

  #sort
  @sorted = $Collator->sort(@not_sorted);

  #compare
  $result = $Collator->cmp($a, $b); # returns 1, 0, or -1.

  # If %tailoring is false (i.e. empty),
  # $Collator should do the default collation.
```

DESCRIPTION

This module is an implementation of Unicode Technical Standard #10 (a.k.a. UTS #10) - Unicode Collation Algorithm (a.k.a. UCA).

Constructor and Tailoring

The new method returns a collator object.

```
   $Collator = Unicode::Collate->new(
      UCA_Version => $UCA_Version,
      alternate => $alternate, # deprecated: use of 'variable' is
recommended.
      backwards => $levelNumber, # or \@levelNumbers
      entry => $element,
      hangul_terminator => $term_primary_weight,
      ignoreName => qr/$ignoreName/,
      ignoreChar => qr/$ignoreChar/,
      katakana_before_hiragana => $bool,
      level => $collationLevel,
      normalization  => $normalization_form,
      overrideCJK => \&overrideCJK,
      overrideHangul => \&overrideHangul,
      preprocess => \&preprocess,
      rearrange => \@charList,
      table => $filename,
      undefName => qr/$undefName/,
      undefChar => qr/$undefChar/,
      upper_before_lower => $bool,
      variable => $variable,
   );
```

UCA_Version

If the tracking version number of UCA is given, behavior of that tracking version is emulated on collating. If omitted, the return value of `UCA_Version()` is used. `UCA_Version()` should return the latest tracking version supported.

The supported tracking version: 8, 9, 11, or 14.

```
UCA       Unicode Standard         DUCET (@version)
--------------------------------------------------
 8              3.1                3.0.1 (3.0.1d9)
 9     3.1 with Corrigendum 3      3.1.1 (3.1.1)
11              4.0                4.0.0 (4.0.0)
14            4.1.0                4.1.0 (4.1.0)
```

Note: Recent UTS #10 renames "Tracking Version" to "Revision."

alternate

-- see 3.2.2 Alternate Weighting, version 8 of UTS #10

For backward compatibility, `alternate` (old name) can be used as an alias for `variable`.

backwards

-- see 3.1.2 French Accents, UTS #10.

```
backwards => $levelNumber or \@levelNumbers
```

Weights in reverse order; ex. level 2 (diacritic ordering) in French. If omitted, forwards at all the levels.

entry

-- see 3.1 Linguistic Features; 3.2.1 File Format, UTS #10.

If the same character (or a sequence of characters) exists in the collation element table through `table`, mapping to collation elements is overrided. If it does not exist, the mapping is defined additionally.

```
    entry => <<'ENTRY', # for DUCET v4.0.0 (allkeys-4.0.0.txt)
0063 0068 ; [.0E6A.0020.0002.0063] # ch
0043 0068 ; [.0E6A.0020.0007.0043] # Ch
0043 0048 ; [.0E6A.0020.0008.0043] # CH
006C 006C ; [.0F4C.0020.0002.006C] # ll
004C 006C ; [.0F4C.0020.0007.004C] # Ll
004C 004C ; [.0F4C.0020.0008.004C] # LL
00F1      ; [.0F7B.0020.0002.00F1] # n-tilde
006E 0303 ; [.0F7B.0020.0002.00F1] # n-tilde
00D1      ; [.0F7B.0020.0008.00D1] # N-tilde
004E 0303 ; [.0F7B.0020.0008.00D1] # N-tilde
ENTRY

    entry => <<'ENTRY', # for DUCET v4.0.0 (allkeys-4.0.0.txt)
00E6 ; [.0E33.0020.0002.00E6][.0E8B.0020.0002.00E6] # ae ligature as
<a><e>
00C6 ; [.0E33.0020.0008.00C6][.0E8B.0020.0008.00C6] # AE ligature as
<A><E>
ENTRY
```

NOTE: The code point in the UCA file format (before `';'`) **must** be a Unicode code point (defined as hexadecimal), but not a native code point. So `0063` must always denote `U+0063`, but not a character of `"\x63"`.

Weighting may vary depending on collation element table. So ensure the weights defined in `entry` will be consistent with those in the collation element table loaded via `table`.

In DUCET v4.0.0, primary weight of `C` is `0E60` and that of `D` is `0E6D`. So setting primary weight of `CH` to `0E6A` (as a value between `0E60` and `0E6D`) makes ordering as `C < CH < D`. Exactly speaking DUCET already has some characters between `C` and `D`: `small capital C` (`U+1D04`) with primary weight `0E64`, `c-hook/C-hook` (`U+0188/U+0187`) with `0E65`, and

`c-curl` (`U+0255`) with `0E69`. Then primary weight `0E6A` for `CH` makes `CH` ordered between `c-curl` and `D`.

hangul_terminator

-- see 7.1.4 Trailing Weights, UTS #10.

If a true value is given (non-zero but should be positive), it will be added as a terminator primary weight to the end of every standard Hangul syllable. Secondary and any higher weights for terminator are set to zero. If the value is false or `hangul_terminator` key does not exist, insertion of terminator weights will not be performed.

Boundaries of Hangul syllables are determined according to conjoining Jamo behavior in *the Unicode Standard* and *HangulSyllableType.txt*.

Implementation Note: (1) For expansion mapping (Unicode character mapped to a sequence of collation elements), a terminator will not be added between collation elements, even if Hangul syllable boundary exists there. Addition of terminator is restricted to the next position to the last collation element.

(2) Non-conjoining Hangul letters (Compatibility Jamo, halfwidth Jamo, and enclosed letters) are not automatically terminated with a terminator primary weight. These characters may need terminator included in a collation element table beforehand.

ignoreChar

ignoreName

-- see 3.2.2 Variable Weighting, UTS #10.

Makes the entry in the table completely ignorable; i.e. as if the weights were zero at all level.

Through `ignoreChar`, any character matching `qr/$ignoreChar/` will be ignored. Through `ignoreName`, any character whose name (given in the `table` file as a comment) matches `qr/$ignoreName/` will be ignored.

E.g. when 'a' and 'e' are ignorable, 'element' is equal to 'lament' (or 'lmnt').

katakana_before_hiragana

-- see 7.3.1 Tertiary Weight Table, UTS #10.

By default, hiragana is before katakana. If the parameter is made true, this is reversed.

NOTE: This parameter simplemindedly assumes that any hiragana/katakana distinctions must occur in level 3, and their weights at level 3 must be same as those mentioned in 7.3.1, UTS #10. If you define your collation elements which violate this requirement, this parameter does not work validly.

level

-- see 4.3 Form Sort Key, UTS #10.

Set the maximum level. Any higher levels than the specified one are ignored.

```
Level 1: alphabetic ordering
Level 2: diacritic ordering
Level 3: case ordering
Level 4: tie-breaking (e.g. in the case when variable is 'shifted')

ex.level => 2,
```

If omitted, the maximum is the 4th.

normalization

-- see 4.1 Normalize, UTS #10.

If specified, strings are normalized before preparation of sort keys (the normalization is executed after preprocess).

A form name `Unicode::Normalize::normalize()` accepts will be applied as `$normalization_form`. Acceptable names include `'NFD'`, `'NFC'`, `'NFKD'`, and `'NFKC'`. See `Unicode::Normalize::normalize()` for detail. If omitted, `'NFD'` is used.

`normalization` is performed after `preprocess` (if defined).

Furthermore, special values, `undef` and `"prenormalized"`, can be used, though they are not concerned with `Unicode::Normalize::normalize()`.

If `undef` (not a string `"undef"`) is passed explicitly as the value for this key, any normalization is not carried out (this may make tailoring easier if any normalization is not desired). Under `(normalization => undef)`, only contiguous contractions are resolved; e.g. even if `A-ring` (and `A-ring-cedilla`) is ordered after `Z`, `A-cedilla-ring` would be primary equal to `A`. In this point, `(normalization => undef, preprocess => sub { NFD(shift) })` **is not** equivalent to `(normalization => 'NFD')`.

In the case of `(normalization => "prenormalized")`, any normalization is not performed, but non-contiguous contractions with combining characters are performed. Therefore `(normalization => 'prenormalized', preprocess => sub { NFD(shift) })` **is** equivalent to `(normalization => 'NFD')`. If source strings are finely prenormalized, `(normalization => 'prenormalized')` may save time for normalization.

Except `(normalization => undef)`, **Unicode::Normalize** is required (see also **CAVEAT**).

overrideCJK

-- see 7.1 Derived Collation Elements, UTS #10.

By default, CJK Unified Ideographs are ordered in Unicode codepoint order but `CJK Unified Ideographs` (if `UCA_Version` is 8 to 11, its range is `U+4E00..U+9FA5`; if `UCA_Version` is 14, its range is `U+4E00..U+9FBB`) are lesser than `CJK Unified Ideographs Extension` (its range is `U+3400..U+4DB5` and `U+20000..U+2A6D6`).

Through `overrideCJK`, ordering of CJK Unified Ideographs can be overrided.

ex. CJK Unified Ideographs in the JIS code point order.

```
overrideCJK => sub {
    my $u = shift;             # get a Unicode codepoint
    my $b = pack('n', $u);     # to UTF-16BE
    my $s = your_unicode_to_sjis_converter($b); # convert
    my $n = unpack('n', $s);   # convert sjis to short
    [ $n, 0x20, 0x2, $u ];     # return the collation element
},
```

ex. ignores all CJK Unified Ideographs.

```
overrideCJK => sub {()}, # CODEREF returning empty list

 # where ->eq("Pe\x{4E00}rl", "Perl") is true
 # as U+4E00 is a CJK Unified Ideograph and to be ignorable.
```

If `undef` is passed explicitly as the value for this key, weights for CJK Unified Ideographs are treated as undefined. But assignment of weight for CJK Unified Ideographs in table or `entry` is still valid.

overrideHangul

-- see 7.1 Derived Collation Elements, UTS #10.

By default, Hangul Syllables are decomposed into Hangul Jamo, even if `(normalization => undef)`. But the mapping of Hangul Syllables may be overrided.

This parameter works like `overrideCJK`, so see there for examples.

If you want to override the mapping of Hangul Syllables, NFD, NFKD, and FCD are not appropriate, since they will decompose Hangul Syllables before overriding.

If `undef` is passed explicitly as the value for this key, weight for Hangul Syllables is treated as undefined without decomposition into Hangul Jamo. But definition of weight for Hangul Syllables in table or `entry` is still valid.

preprocess

-- see 5.1 Preprocessing, UTS #10.

If specified, the coderef is used to preprocess before the formation of sort keys.

ex. dropping English articles, such as "a" or "the". Then, "the pen" is before "a pencil".

```
preprocess => sub {
      my $str = shift;
      $str =~ s/\b(?:an?|the)\s+//gi;
      return $str;
   },
```

`preprocess` is performed before `normalization` (if defined).

rearrange

-- see 3.1.3 Rearrangement, UTS #10.

Characters that are not coded in logical order and to be rearranged. If `UCA_Version` is equal to or lesser than 11, default is:

```
rearrange => [ 0x0E40..0x0E44, 0x0EC0..0x0EC4 ],
```

If you want to disallow any rearrangement, pass `undef` or `[]` (a reference to empty list) as the value for this key.

If `UCA_Version` is equal to 14, default is `[]` (i.e. no rearrangement).

According to the version 9 of UCA, this parameter shall not be used; but it is not warned at present.

table

-- see 3.2 Default Unicode Collation Element Table, UTS #10.

You can use another collation element table if desired.

The table file should locate in the *Unicode/Collate* directory on `@INC`. Say, if the filename is *Foo.txt*, the table file is searched as *Unicode/Collate/Foo.txt* in `@INC`.

By default, *allkeys.txt* (as the filename of DUCET) is used. If you will prepare your own table file, any name other than *allkeys.txt* may be better to avoid namespace conflict.

If `undef` is passed explicitly as the value for this key, no file is read (but you can define collation elements via `entry`).

A typical way to define a collation element table without any file of table:

```
   $onlyABC = Unicode::Collate->new(
       table => undef,
       entry => << 'ENTRIES',
0061 ; [.0101.0020.0002.0061] # LATIN SMALL LETTER A
0041 ; [.0101.0020.0008.0041] # LATIN CAPITAL LETTER A
0062 ; [.0102.0020.0002.0062] # LATIN SMALL LETTER B
0042 ; [.0102.0020.0008.0042] # LATIN CAPITAL LETTER B
0063 ; [.0103.0020.0002.0063] # LATIN SMALL LETTER C
0043 ; [.0103.0020.0008.0043] # LATIN CAPITAL LETTER C
ENTRIES
    );
```

If `ignoreName` or `undefName` is used, character names should be specified as a comment (following #) on each line.

undefChar

undefName

-- see 6.3.4 Reducing the Repertoire, UTS #10.

Undefines the collation element as if it were unassigned in the table. This reduces the size of the table. If an unassigned character appears in the string to be collated, the sort key is made from its codepoint as a single-character collation element, as it is greater than any other assigned collation elements (in the codepoint order among the unassigned characters). But, it'd be better to ignore characters unfamiliar to you and maybe never used.

Through `undefChar`, any character matching `qr/$undefChar/` will be undefined. Through `undefName`, any character whose name (given in the `table` file as a comment) matches `qr/$undefName/` will be undefined.

ex. Collation weights for beyond-BMP characters are not stored in object:

```
undefChar => qr/[^\0-\x{fffd}]/,
```

upper_before_lower

-- see 6.6 Case Comparisons, UTS #10.

By default, lowercase is before uppercase. If the parameter is made true, this is reversed.

NOTE: This parameter simplemindedly assumes that any lowercase/uppercase distinctions must occur in level 3, and their weights at level 3 must be same as those mentioned in 7.3.1, UTS #10. If you define your collation elements which differs from this requirement, this parameter doesn't work validly.

variable

-- see 3.2.2 Variable Weighting, UTS #10.

This key allows to variable weighting for variable collation elements, which are marked with an ASTERISK in the table (NOTE: Many punction marks and symbols are variable in *allkeys.txt*).

```
   variable => 'blanked', 'non-ignorable', 'shifted', or
'shift-trimmed'.
```

These names are case-insensitive. By default (if specification is omitted), 'shifted' is adopted.

```
   'Blanked'        Variable elements are made ignorable at levels 1
through 3;
                    considered at the 4th level.

   'Non-Ignorable'  Variable elements are not reset to ignorable.

   'Shifted'        Variable elements are made ignorable at levels 1
through 3
                    their level 4 weight is replaced by the old level
 1 weight.
                    Level 4 weight for Non-Variable elements is
0xFFFF.

   'Shift-Trimmed'  Same as 'shifted', but all FFFF's at the 4th
level
                    are trimmed.
```

Methods for Collation

`@sorted = $Collator->sort(@not_sorted)`

Sorts a list of strings.

`$result = $Collator->cmp($a, $b)`

Returns 1 (when $a is greater than $b) or 0 (when $a is equal to $b) or -1 (when $a is lesser than $b).

`$result = $Collator->eq($a, $b)`

`$result = $Collator->ne($a, $b)`

`$result = $Collator->lt($a, $b)`

`$result = $Collator->le($a, $b)`

`$result = $Collator->gt($a, $b)`

`$result = $Collator->ge($a, $b)`

They works like the same name operators as theirs.

```
eq : whether $a is equal to $b.
ne : whether $a is not equal to $b.
lt : whether $a is lesser than $b.
le : whether $a is lesser than $b or equal to $b.
gt : whether $a is greater than $b.
ge : whether $a is greater than $b or equal to $b.
```

`$sortKey = $Collator->getSortKey($string)`

-- see 4.3 Form Sort Key, UTS #10.

Returns a sort key.

You compare the sort keys using a binary comparison and get the result of the comparison of the strings using UCA.

```
$Collator->getSortKey($a) cmp $Collator->getSortKey($b)

   is equivalent to

$Collator->cmp($a, $b)
```

`$sortKeyForm = $Collator->viewSortKey($string)`

Converts a sorting key into its representation form. If `UCA_Version` is 8, the output is slightly different.

```
use Unicode::Collate;
my $c = Unicode::Collate->new();
print $c->viewSortKey("Perl"),"\n";

# output:
# [0B67 0A65 0B7F 0B03 | 0020 0020 0020 0020 | 0008 0002 0002 0002
 | FFFF FFFF FFFF FFFF]
#  Level 1              Level 2              Level 3
            Level 4
```

Methods for Searching

DISCLAIMER: If `preprocess` or `normalization` parameter is true for `$Collator`, calling these methods (`index`, `match`, `gmatch`, `subst`, `gsubst`) is croaked, as the position and the length might differ from those on the specified string. (And `rearrange` and `hangul_terminator` parameters are

neglected.)

The `match`, `gmatch`, `subst`, `gsubst` methods work like `m//`, `m//g`, `s///`, `s///g`, respectively, but they are not aware of any pattern, but only a literal substring.

```
$position = $Collator->index($string, $substring[, $position])
($position, $length) = $Collator->index($string, $substring[, $position])
```

If `$substring` matches a part of `$string`, returns the position of the first occurrence of the matching part in scalar context; in list context, returns a two-element list of the position and the length of the matching part.

If `$substring` does not match any part of `$string`, returns `-1` in scalar context and an empty list in list context.

e.g. you say

```
  my $Collator = Unicode::Collate->new( normalization => undef, level
 => 1 );
                                        # (normalization => undef) is
REQUIRED.
  my $str = "Ich muß studieren Perl.";
  my $sub = "MÜSS";
  my $match;
  if (my($pos,$len) = $Collator->index($str, $sub)) {
      $match = substr($str, $pos, $len);
  }
```

and get `"muß"` in `$match` since `"muß"` is primary equal to `"MÜSS"`.

```
$match_ref = $Collator->match($string, $substring)
($match)   = $Collator->match($string, $substring)
```

If `$substring` matches a part of `$string`, in scalar context, returns **a reference to** the first occurrence of the matching part (`$match_ref` is always true if matches, since every reference is **true**); in list context, returns the first occurrence of the matching part.

If `$substring` does not match any part of `$string`, returns `undef` in scalar context and an empty list in list context.

e.g.

```
    if ($match_ref = $Collator->match($str, $sub)) { # scalar context
 print "matches [$$match_ref].\n";
    } else {
 print "doesn't match.\n";
    }

     or

    if (($match) = $Collator->match($str, $sub)) { # list context
 print "matches [$match].\n";
    } else {
 print "doesn't match.\n";
    }
```

```
@match = $Collator->gmatch($string, $substring)
```

If `$substring` matches a part of `$string`, returns all the matching parts (or matching count in scalar context).

If `$substring` does not match any part of `$string`, returns an empty list.

`$count = $Collator->subst($string, $substring, $replacement)`

If `$substring` matches a part of `$string`, the first occurrence of the matching part is replaced by `$replacement` (`$string` is modified) and return `$count` (always equals to `1`).

`$replacement` can be a `CODEREF`, taking the matching part as an argument, and returning a string to replace the matching part (a bit similar to `s/(..)/$coderef->($1)/e`).

`$count = $Collator->gsubst($string, $substring, $replacement)`

If `$substring` matches a part of `$string`, all the occurrences of the matching part is replaced by `$replacement` (`$string` is modified) and return `$count`.

`$replacement` can be a `CODEREF`, taking the matching part as an argument, and returning a string to replace the matching part (a bit similar to `s/(..)/$coderef->($1)/eg`).

e.g.

```
  my $Collator = Unicode::Collate->new( normalization => undef, level
 => 1 );
                                        # (normalization => undef) is
REQUIRED.
  my $str = "Camel donkey zebra came\x{301}l CAMEL horse
cAm\0E\0L...";
  $Collator->gsubst($str, "camel", sub { "<b>$_[0]</b>" });

  # now $str is "<b>Camel</b> donkey zebra <b>came\x{301}l</b>
<b>CAMEL</b> horse <b>cAm\0E\0L</b>...";
  # i.e., all the camels are made bold-faced.
```

Other Methods

`%old_tailoring = $Collator->change(%new_tailoring)`

Change the value of specified keys and returns the changed part.

```
    $Collator = Unicode::Collate->new(level => 4);

    $Collator->eq("perl", "PERL"); # false

    %old = $Collator->change(level => 2); # returns (level => 4).

    $Collator->eq("perl", "PERL"); # true

    $Collator->change(%old); # returns (level => 2).

    $Collator->eq("perl", "PERL"); # false
```

Not all `(key,value)`s are allowed to be changed. See also `@Unicode::Collate::ChangeOK` and `@Unicode::Collate::ChangeNG`.

In the scalar context, returns the modified collator (but it is **not** a clone from the original).

```
    $Collator->change(level => 2)->eq("perl", "PERL"); # true

    $Collator->eq("perl", "PERL"); # true; now max level is 2nd.

    $Collator->change(level => 4)->eq("perl", "PERL"); # false
```

`$version = $Collator->version()`

Returns the version number (a string) of the Unicode Standard which the `table` file used by the collator object is based on. If the table does not include a version line (starting with `@version`), returns `"unknown"`.

`UCA_Version()`

Returns the tracking version number of UTS #10 this module consults.

`Base_Unicode_Version()`

Returns the version number of UTS #10 this module consults.

EXPORT

No method will be exported.

INSTALL

Though this module can be used without any `table` file, to use this module easily, it is recommended to install a table file in the UCA format, by copying it under the directory <a place in @INC>/Unicode/Collate.

The most preferable one is "The Default Unicode Collation Element Table" (aka DUCET), available from the Unicode Consortium's website:

```
http://www.unicode.org/Public/UCA/

http://www.unicode.org/Public/UCA/latest/allkeys.txt (latest version)
```

If DUCET is not installed, it is recommended to copy the file from http://www.unicode.org/Public/UCA/latest/allkeys.txt to <a place in @INC>/Unicode/Collate/allkeys.txt manually.

CAVEATS

Normalization

Use of the `normalization` parameter requires the **Unicode::Normalize** module (see *Unicode::Normalize*).

If you need not it (say, in the case when you need not handle any combining characters), assign `normalization => undef` explicitly.

-- see 6.5 Avoiding Normalization, UTS #10.

Conformance Test

The Conformance Test for the UCA is available under *http://www.unicode.org/Public/UCA/*.

For *CollationTest_SHIFTED.txt*, a collator via `Unicode::Collate->new( )` should be used; for *CollationTest_NON_IGNORABLE.txt*, a collator via `Unicode::Collate->new(variable => "non-ignorable", level => 3)`.

Unicode::Normalize is required to try The Conformance Test.

AUTHOR, COPYRIGHT AND LICENSE

SEE ALSO

Unicode Collation Algorithm - UTS #10

http://www.unicode.org/reports/tr10/

The Default Unicode Collation Element Table (DUCET)

http://www.unicode.org/Public/UCA/latest/allkeys.txt

The conformance test for the UCA

http://www.unicode.org/Public/UCA/latest/CollationTest.html

http://www.unicode.org/Public/UCA/latest/CollationTest.zip

Hangul Syllable Type

http://www.unicode.org/Public/UNIDATA/HangulSyllableType.txt

Unicode Normalization Forms - UAX #15

http://www.unicode.org/reports/tr15/