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.
```

Note: Strings in `@not_sorted`, $a and $b are interpreted according to Perl's Unicode support. See *perlunicode*, *perluniintro*, *perlunitut*, *perlunifaq*, *utf8*. Otherwise you can use `preprocess` or should decode them before.

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. If new() is called with no parameters, the collator should do the default collation.

```
$Collator = Unicode::Collate->new(
   UCA_Version => $UCA_Version,
   alternate => $alternate, # alias for 'variable'
   backwards => $levelNumber, # or \@levelNumbers
   entry => $element,
   hangul_terminator => $term_primary_weight,
   highestFFFF => $bool,
   identical => $bool,
   ignoreName => qr/$ignoreName/,
   ignoreChar => qr/$ignoreChar/,
   ignore_level2 => $bool,
   katakana_before_hiragana => $bool,
   level => $collationLevel,
   long_contraction => $bool,
   minimalFFFE => $bool,
   normalization  => $normalization_form,
   overrideCJK => \&overrideCJK,
   overrideHangul => \&overrideHangul,
   preprocess => \&preprocess,
   rearrange => \@charList,
   rewrite => \&rewrite,
   suppress => \@charList,
   table => $filename,
   undefName => qr/$undefName/,
   undefChar => qr/$undefChar/,
   upper_before_lower => $bool,
   variable => $variable,
 );
```

UCA_Version

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

The following revisions are supported. The default is 32.

```
    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)
     16                5.0              5.0.0 (5.0.0)
     18                5.1.0            5.1.0 (5.1.0)
     20                5.2.0            5.2.0 (5.2.0)
     22                6.0.0            6.0.0 (6.0.0)
     24                6.1.0            6.1.0 (6.1.0)
     26                6.2.0            6.2.0 (6.2.0)
     28                6.3.0            6.3.0 (6.3.0)
     30                7.0.0            7.0.0 (7.0.0)
     32                8.0.0            8.0.0 (8.0.0)
```

* See below for `long_contraction` with `UCA_Version` 22 and 24.

* Noncharacters (e.g. U+FFFF) are not ignored, and can be overridden since `UCA_Version` 22.

* Out-of-range codepoints (greater than U+10FFFF) are not ignored, and can be overridden since `UCA_Version` 22.

* Fully ignorable characters were ignored, and would not interrupt contractions with `UCA_Version` 9 and 11.

* Treatment of ignorables after variables and some behaviors were changed at `UCA_Version` 9.

* Characters regarded as CJK unified ideographs (cf. `overrideCJK`) depend on `UCA_Version`.

* Many hangul jamo are assigned at `UCA_Version` 20, that will affect `hangul_terminator`.

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 3.4 Backward Accents, UTS #10.

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

Weights in reverse order; ex. level 2 (diacritic ordering) in French. If omitted (or `$levelNumber` is `undef` or `\@levelNumbers` is `[]`), forwards at all the levels.

entry

-- see 5 Tailoring; 9.1 Allkeys 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 overridden. 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.

highestFFFF

> -- see 2.4 Tailored noncharacter weights, UTS #35 (LDML) Part 5: Collation.

> If the parameter is made true, `U+FFFF` has a highest primary weight. When a boolean of `$coll->ge($str, "abc")` and `$coll->le($str, "abc\x{FFFF}")` is true, it is expected that `$str` begins with `"abc"`, or another primary equivalent. `$str` may be `"abcd"`, `"abc012"`, but should not include `U+FFFF` such as `"abc\x{FFFF}xyz"`.

> `$coll->le($str, "abc\x{FFFF}")` works like `$coll->lt($str, "abd")` almost, but the latter has a problem that you should know which letter is next to `c`. For a certain language

where `ch` as the next letter, `"abch"` is greater than `"abc\x{FFFF}"`, but less than `"abd"`.

Note: This is equivalent to `(entry => 'FFFF ; [.FFFE.0020.0005.FFFF]')`. Any other character than `U+FFFF` can be tailored by `entry`.

identical

-- see A.3 Deterministic Comparison, UTS #10.

By default, strings whose weights are equal should be equal, even though their code points are not equal. Completely ignorable characters are ignored.

If the parameter is made true, a final, tie-breaking level is used. If no difference of weights is found after the comparison through all the level specified by `level`, the comparison with code points will be performed. For the tie-breaking comparison, the sort key has code points of the original string appended. Completely ignorable characters are not ignored.

If `preprocess` and/or `normalization` is applied, the code points of the string after them (in NFD by default) are used.

ignoreChar

ignoreName

-- see 3.6 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').

ignore_level2

-- see 5.1 Parametric Tailoring, UTS #10.

By default, case-sensitive comparison (that is level 3 difference) won't ignore accents (that is level 2 difference).

If the parameter is made true, accents (and other primary ignorable characters) are ignored, even though cases are taken into account.

NOTE: `level` should be 3 or greater.

katakana_before_hiragana

-- see 7.2 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.

NOTE: The DUCET includes weights over 0xFFFF at the 4th level. But this module only uses weights within 0xFFFF. When `variable` is 'blanked' or 'non-ignorable' (other than 'shifted' and 'shift-trimmed'), the level 4 may be unreliable.

See also `identical`.

long_contraction

-- see 3.8.2 Well-Formedness of the DUCET, 4.2 Produce Array, UTS #10.

If the parameter is made true, for a contraction with three or more characters (here nicknamed "long contraction"), initial substrings will be handled. For example, a contraction ABC, where A is a starter, and B and C are non-starters (character with non-zero combining character class), will be detected even if there is not AB as a contraction.

Default: Usually false. If `UCA_Version` is 22 or 24, and the value of `long_contraction` is not specified in `new()`, a true value is set implicitly. This is a workaround to pass Conformance Tests for Unicode 6.0.0 and 6.1.0.

`change()` handles `long_contraction` explicitly only. If `long_contraction` is not specified in `change()`, even though `UCA_Version` is changed, `long_contraction` will not be changed.

Limitation: Scanning non-starters is one-way (no back tracking). If AB is found but not ABC is not found, other long contraction where the first character is A and the second is not B may not be found.

Under `(normalization => undef)`, detection step of discontiguous contractions will be skipped.

Note: The following contractions in DUCET are not considered in steps S2.1.1 to S2.1.3, where they are discontiguous.

```
0FB2 0F71 0F80 (TIBETAN VOWEL SIGN VOCALIC RR)
0FB3 0F71 0F80 (TIBETAN VOWEL SIGN VOCALIC LL)
```

For example `TIBETAN VOWEL SIGN VOCALIC RR` with `COMBINING TILDE OVERLAY` (`U+0344`) is `0FB2 0344 0F71 0F80` in NFD. In this case `0FB2 0F80` (`TIBETAN VOWEL SIGN VOCALIC R`) is detected, instead of `0FB2 0F71 0F80`. Inserted `0344` makes `0FB2 0F71 0F80` discontiguous and lack of contraction `0FB2 0F71` prohibits `0FB2 0F71 0F80` from being detected.

minimalFFFE

-- see 1.1.1 U+FFFE, UTS #35 (LDML) Part 5: Collation.

If the parameter is made true, `U+FFFE` has a minimal primary weight. The comparison between `"$a1\x{FFFE}$a2"` and `"$b1\x{FFFE}$b2"` first compares `$a1` and `$b1` at level 1, and then `$a2` and `$b2` at level 1, as followed.

```
"ab\x{FFFE}a"
"Ab\x{FFFE}a"
"ab\x{FFFE}c"
"Ab\x{FFFE}c"
"ab\x{FFFE}xyz"
"abc\x{FFFE}def"
"abc\x{FFFE}xYz"
"aBc\x{FFFE}xyz"
"abcX\x{FFFE}def"
"abcx\x{FFFE}xyz"
"b\x{FFFE}aaa"
"bbb\x{FFFE}a"
```

Note: This is equivalent to `(entry => 'FFFE ; [.0001.0020.0005.FFFE]')`. Any other character than `U+FFFE` can be tailored by `entry`.

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 discontiguous 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 those in the CJK Unified Ideographs block are less than those in the CJK Unified Ideographs Extension A etc.

```
In the CJK Unified Ideographs block:
U+4E00..U+9FA5 if UCA_Version is 8, 9 or 11.
U+4E00..U+9FBB if UCA_Version is 14 or 16.
U+4E00..U+9FC3 if UCA_Version is 18.
U+4E00..U+9FCB if UCA_Version is 20 or 22.
U+4E00..U+9FCC if UCA_Version is 24 to 30.
U+4E00..U+9FD5 if UCA_Version is 32.

In the CJK Unified Ideographs Extension blocks:
Ext.A (U+3400..U+4DB5) and Ext.B (U+20000..U+2A6D6) in any
UCA_Version.
Ext.C (U+2A700..U+2B734) if UCA_Version is 20 or later.
Ext.D (U+2B740..U+2B81D) if UCA_Version is 22 or later.
Ext.E (U+2B820..U+2CEA1) if UCA_Version is 32.
```

Through `overrideCJK`, ordering of CJK unified ideographs (including extensions) can be overridden.

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
    },
```

The return value may be an arrayref of 1st to 4th weights as shown above. The return value may be an integer as the primary weight as shown below. If `undef` is returned, the default derived collation element will be used.

```
  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
      return $n;                    # return the primary weight
  },
```

The return value may be a list containing zero or more of an arrayref, an integer, or `undef`.

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 a false value (including `undef`) is passed, `overrideCJK` has no effect. `$Collator->change(overrideCJK => 0)` resets the old one.

But assignment of weight for CJK unified ideographs in `table` or `entry` is still valid. If `undef` is passed explicitly as the value for this key, weights for CJK unified ideographs are treated as undefined. However when `UCA_Version` > 8, `(overrideCJK => undef)` has no special meaning.

Note: In addition to them, 12 CJK compatibility ideographs (`U+FA0E`, `U+FA0F`, `U+FA11`, `U+FA13`, `U+FA14`, `U+FA1F`, `U+FA21`, `U+FA23`, `U+FA24`, `U+FA27`, `U+FA28`, `U+FA29`) are also treated as CJK unified ideographs. But they can't be overridden via `overrideCJK` when you use DUCET, as the table includes weights for them. `table` or `entry` has priority over `overrideCJK`.

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 overridden.

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

If you want to override the mapping of Hangul syllables, NFD and NFKD are not appropriate, since NFD and NFKD will decompose Hangul syllables before overriding. FCD may decompose Hangul syllables as the case may be.

If a false value (but not `undef`) is passed, `overrideHangul` has no effect. `$Collator->change(overrideHangul => 0)` resets the old one.

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.

overrideOut

-- see 7.1.1 Handling Ill-Formed Code Unit Sequences, UTS #10.

Perl seems to allow out-of-range values (greater than 0x10FFFF). By default, out-of-range values are replaced with `U+FFFD` (REPLACEMENT CHARACTER) when `UCA_Version` >=

22, or ignored when `UCA_Version` <= 20.

When `UCA_Version` >= 22, the weights of out-of-range values can be overridden. Though `table` or `entry` are available for them, out-of-range values are too many.

`overrideOut` can perform it algorithmically. This parameter works like `overrideCJK`, so see there for examples.

ex. ignores all out-of-range values.

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

If a false value (including `undef`) is passed, `overrideOut` has no effect. `$Collator->change(overrideOut => 0)` resets the old one.

NOTE ABOUT U+FFFD:

UCA recommends that out-of-range values should not be ignored for security reasons. Say, `"pe\x{110000}rl"` should not be equal to `"perl"`. However, U+FFFD is wrongly mapped to a variable collation element in DUCET for Unicode 6.0.0 to 6.2.0, that means out-of-range values will be ignored when `variable` isn't `Non-ignorable`.

The mapping of U+FFFD is corrected in Unicode 6.3.0. see *http://www.unicode.org/reports/tr10/tr10-28.html#Trailing_Weights* (7.1.4 Trailing Weights). Such a correction is reproduced by this.

```
overrideOut => sub { 0xFFFD }, # CODEREF returning a very large
integer
```

This workaround is unnecessary since Unicode 6.3.0.

preprocess

-- see 5.4 Preprocessing, UTS #10.

If specified, the coderef is used to preprocess each string 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).

ex. decoding strings in a legacy encoding such as shift-jis:

```
$sjis_collator = Unicode::Collate->new(
    preprocess => \&your_shiftjis_to_unicode_decoder,
);
@result = $sjis_collator->sort(@shiftjis_strings);
```

Note: Strings returned from the coderef will be interpreted according to Perl's Unicode support. See *perlunicode*, *perluniintro*, *perlunitut*, *perlunifaq*, *utf8*.

rearrange

-- see 3.5 Rearrangement, UTS #10.

Characters that are not coded in logical order and to be rearranged. If `UCA_Version` is equal to or less 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 or greater than 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.

rewrite

If specified, the coderef is used to rewrite lines in `table` or `entry`. The coderef will get each line, and then should return a rewritten line according to the UCA file format. If the coderef returns an empty line, the line will be skipped.

e.g. any primary ignorable characters into tertiary ignorable:

```
rewrite => sub {
    my $line = shift;
    $line =~ s/\[\.0000\..{4}\..{4}\./[.0000.0000.0000./g;
    return $line;
},
```

This example shows rewriting weights. `rewrite` is allowed to affect code points, weights, and the name.

NOTE: `table` is available to use another table file; preparing a modified table once would be more efficient than rewriting lines on reading an unmodified table every time.

suppress

-- see 3.12 Special-Purpose Commands, UTS #35 (LDML) Part 5: Collation.

Contractions beginning with the specified characters are suppressed, even if those contractions are defined in `table`.

An example for Russian and some languages using the Cyrillic script:

```
suppress => [0x0400..0x0417, 0x041A..0x0437, 0x043A..0x045F],
```

where 0x0400 stands for `U+0400`, CYRILLIC CAPITAL LETTER IE WITH GRAVE.

NOTE: Contractions via `entry` will not be suppressed.

table

-- see 3.8 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.

NOTE: When XSUB is used, the DUCET is compiled on building this module, and it may save time at the run time. Explicit saying `(table => 'allkeys.txt')`, or using another table, or using `ignoreChar`, `ignoreName`, `undefChar`, `undefName` or `rewrite` will prevent this module from using the compiled DUCET.

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 6.3.3 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 3.6 Variable Weighting, UTS #10.

This key allows for variable weighting of variable collation elements, which are marked with an ASTERISK in the table (NOTE: Many punctuation 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 less 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 less than $b.
            le : whether $a is less 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

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.

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.

`rearrange` and `hangul_terminator` parameters are neglected. `katakana_before_hiragana` and `upper_before_lower` don't affect matching and searching, as it doesn't matter whether greater or less.

`$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. when the content of `$str` is `"Ich muß studieren Perl."`, you say the following where `$sub` is `"MüSS"`,

```
  my $Collator = Unicode::Collate->new( normalization => undef, level
  => 1 );
                                    # (normalization => undef) is
REQUIRED.
  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 `$count` (always equals to `1`) is returned.

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

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

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

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

   Examples: levels and ignore_level2 - what does camel match?
```

```
----------------------------------------------------------------------
------
    level  ignore_level2  |  camel  Camel  came\x{301}l  c-a-m-e-l
cam\0e\0l

----------------------|-----------------------------------------------
------
     1       false      |   yes    yes      yes          yes
yes
     2       false      |   yes    yes      no           yes
yes
     3       false      |   yes    no       no           yes
yes
     4       false      |   yes    no       no           no
yes


----------------------|-----------------------------------------------
------
     1       true       |   yes    yes      yes          yes
yes
     2       true       |   yes    yes      yes          yes
yes
```

```
     3        true    |   yes    no      yes         yes
  yes
     4        true    |   yes    no      yes         no
  yes


  ------------------------------------------------------------------
  ------
     note: if variable => non-ignorable, camel doesn't match c-a-m-e-l
           at any level.
```

Other Methods

> `%old_tailoring = $Collator->change(%new_tailoring)`
>
> `$modified_collator = $Collator->change(%new_tailoring)`
>
>> Changes 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 revision number of UTS #10 this module consults, that should correspond with the DUCET incorporated.

> `Base_Unicode_Version()`
>
>> Returns the version number of UTS #10 this module consults, that should correspond with the DUCET incorporated.

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)`.
>
> If `UCA_Version` is 26 or later, the `identical` level is preferred; `Unicode::Collate->new(identical => 1)` and `Unicode::Collate->new(identical => 1, variable => "non-ignorable", level => 3)` should be used.
>
> **Unicode::Normalize is required to try The Conformance Test.**

AUTHOR, COPYRIGHT AND LICENSE

The Unicode::Collate module for perl was written by SADAHIRO Tomoyuki, <SADAHIRO@cpan.org>. This module is Copyright(C) 2001-2016, SADAHIRO Tomoyuki. Japan. All rights reserved.

This module is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

The file Unicode/Collate/allkeys.txt was copied verbatim from *http://www.unicode.org/Public/UCA/8.0.0/allkeys.txt*. For this file, Copyright (c) 2001-2015 Unicode, Inc.; distributed under the Terms of Use in *http://www.unicode.org/terms_of_use.html*

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/

Unicode Locale Data Markup Language (LDML) - UTS #35

http://www.unicode.org/reports/tr35/