NAME

Unicode::Normalize - Unicode Normalization Forms

SYNOPSIS

(1) using function names exported by default:

```
use Unicode::Normalize;

$NFD_string  = NFD($string);  # Normalization Form D
$NFC_string  = NFC($string);  # Normalization Form C
$NFKD_string = NFKD($string); # Normalization Form KD
$NFKC_string = NFKC($string); # Normalization Form KC
```

(2) using function names exported on request:

```
use Unicode::Normalize 'normalize';

$NFD_string  = normalize('D',  $string);  # Normalization Form D
$NFC_string  = normalize('C',  $string);  # Normalization Form C
$NFKD_string = normalize('KD', $string);  # Normalization Form KD
$NFKC_string = normalize('KC', $string);  # Normalization Form KC
```

DESCRIPTION

Parameters:

`$string` is used as a string under character semantics (see *perlunicode*).

`$code_point` should be an unsigned integer representing a Unicode code point.

Note: Between XSUB and pure Perl, there is an incompatibility about the interpretation of `$code_point` as a decimal number. XSUB converts `$code_point` to an unsigned integer, but pure Perl does not. Do not use a floating point nor a negative sign in `$code_point`.

Normalization Forms

`$NFD_string = NFD($string)`

It returns the Normalization Form D (formed by canonical decomposition).

`$NFC_string = NFC($string)`

It returns the Normalization Form C (formed by canonical decomposition followed by canonical composition).

`$NFKD_string = NFKD($string)`

It returns the Normalization Form KD (formed by compatibility decomposition).

`$NFKC_string = NFKC($string)`

It returns the Normalization Form KC (formed by compatibility decomposition followed by **canonical** composition).

`$FCD_string = FCD($string)`

If the given string is in FCD ("Fast C or D" form; cf. UTN #5), it returns the string without modification; otherwise it returns an FCD string.

Note: FCD is not always unique, then plural forms may be equivalent each other. `FCD()` will return one of these equivalent forms.

`$FCC_string = FCC($string)`

It returns the FCC form ("Fast C Contiguous"; cf. UTN #5).

Note: FCC is unique, as well as four normalization forms (NF*).

```
$normalized_string = normalize($form_name, $string)
```

It returns the normalization form of `$form_name`.

As `$form_name`, one of the following names must be given.

```
  'C'  or 'NFC'  for Normalization Form C  (UAX #15)
  'D'  or 'NFD'  for Normalization Form D  (UAX #15)
  'KC' or 'NFKC' for Normalization Form KC (UAX #15)
  'KD' or 'NFKD' for Normalization Form KD (UAX #15)

  'FCD'          for "Fast C or D" Form  (UTN #5)
  'FCC'          for "Fast C Contiguous" (UTN #5)
```

Decomposition and Composition

```
$decomposed_string = decompose($string [, $useCompatMapping])
```

It returns the concatenation of the decomposition of each character in the string.

If the second parameter (a boolean) is omitted or false, the decomposition is canonical decomposition; if the second parameter (a boolean) is true, the decomposition is compatibility decomposition.

The string returned is not always in NFD/NFKD. Reordering may be required.

```
    $NFD_string  = reorder(decompose($string));       # eq. to NFD()
    $NFKD_string = reorder(decompose($string, TRUE)); # eq. to NFKD()
```

```
$reordered_string = reorder($string)
```

It returns the result of reordering the combining characters according to Canonical Ordering Behavior.

For example, when you have a list of NFD/NFKD strings, you can get the concatenated NFD/NFKD string from them, by saying

```
    $concat_NFD  = reorder(join '', @NFD_strings);
    $concat_NFKD = reorder(join '', @NFKD_strings);
```

```
$composed_string = compose($string)
```

It returns the result of canonical composition without applying any decomposition.

For example, when you have a NFD/NFKD string, you can get its NFC/NFKC string, by saying

```
    $NFC_string  = compose($NFD_string);
    $NFKC_string = compose($NFKD_string);
```

```
($processed, $unprocessed) = splitOnLastStarter($normalized)
```

It returns two strings: the first one, `$processed`, is a part before the last starter, and the second one, `$unprocessed` is another part after the first part. A starter is a character having a combining class of zero (see UAX #15).

Note that `$processed` may be empty (when `$normalized` contains no starter or starts with the last starter), and then `$unprocessed` should be equal to the entire `$normalized`.

When you have a `$normalized` string and an `$unnormalized` string following it, a simple concatenation is wrong:

```
    $concat = $normalized . normalize($form, $unnormalized); # wrong!
```

Instead of it, do like this:

```
    ($processed, $unprocessed) = splitOnLastStarter($normalized);
     $concat = $processed . normalize($form,
$unprocessed.$unnormalized);
```

`splitOnLastStarter()` should be called with a pre-normalized parameter `$normalized`, that is in the same form as `$form` you want.

If you have an array of `@string` that should be concatenated and then normalized, you can do like this:

```
my $result = "";
my $unproc = "";
foreach my $str (@string) {
    $unproc .= $str;
    my $n = normalize($form, $unproc);
    my($p, $u) = splitOnLastStarter($n);
    $result .= $p;
    $unproc  = $u;
}
$result .= $unproc;
# instead of normalize($form, join('', @string))
```

`$processed = normalize_partial($form, $unprocessed)`

A wrapper for the combination of `normalize()` and `splitOnLastStarter()`. Note that `$unprocessed` will be modified as a side-effect.

If you have an array of `@string` that should be concatenated and then normalized, you can do like this:

```
my $result = "";
my $unproc = "";
foreach my $str (@string) {
    $unproc .= $str;
    $result .= normalize_partial($form, $unproc);
}
$result .= $unproc;
# instead of normalize($form, join('', @string))
```

`$processed = NFD_partial($unprocessed)`

It does like `normalize_partial('NFD', $unprocessed)`. Note that `$unprocessed` will be modified as a side-effect.

`$processed = NFC_partial($unprocessed)`

It does like `normalize_partial('NFC', $unprocessed)`. Note that `$unprocessed` will be modified as a side-effect.

`$processed = NFKD_partial($unprocessed)`

It does like `normalize_partial('NFKD', $unprocessed)`. Note that `$unprocessed` will be modified as a side-effect.

`$processed = NFKC_partial($unprocessed)`

It does like `normalize_partial('NFKC', $unprocessed)`. Note that `$unprocessed` will be modified as a side-effect.

Quick Check

(see Annex 8, UAX #15; and *DerivedNormalizationProps.txt*)

The following functions check whether the string is in that normalization form.

The result returned will be one of the following:

```
    YES     The string is in that normalization form.
    NO      The string is not in that normalization form.
    MAYBE   Dubious. Maybe yes, maybe no.
```

`$result = checkNFD($string)`

It returns true (1) if YES; false (`empty string`) if NO.

`$result = checkNFC($string)`

It returns true (1) if YES; false (`empty string`) if NO; `undef` if MAYBE.

`$result = checkNFKD($string)`

It returns true (1) if YES; false (`empty string`) if NO.

`$result = checkNFKC($string)`

It returns true (1) if YES; false (`empty string`) if NO; `undef` if MAYBE.

`$result = checkFCD($string)`

It returns true (1) if YES; false (`empty string`) if NO.

`$result = checkFCC($string)`

It returns true (1) if YES; false (`empty string`) if NO; `undef` if MAYBE.

Note: If a string is not in FCD, it must not be in FCC. So `checkFCC($not_FCD_string)` should return NO.

`$result = check($form_name, $string)`

It returns true (1) if YES; false (`empty string`) if NO; `undef` if MAYBE.

As `$form_name`, one of the following names must be given.

```
  'C'  or 'NFC'  for Normalization Form C  (UAX #15)
  'D'  or 'NFD'  for Normalization Form D  (UAX #15)
  'KC' or 'NFKC' for Normalization Form KC (UAX #15)
  'KD' or 'NFKD' for Normalization Form KD (UAX #15)

  'FCD'          for "Fast C or D" Form  (UTN #5)
  'FCC'          for "Fast C Contiguous" (UTN #5)
```

Note

In the cases of NFD, NFKD, and FCD, the answer must be either YES or NO. The answer MAYBE may be returned in the cases of NFC, NFKC, and FCC.

A MAYBE string should contain at least one combining character or the like. For example, `COMBINING ACUTE ACCENT` has the MAYBE_NFC/MAYBE_NFKC property.

Both `checkNFC("A\N{COMBINING ACUTE ACCENT}")` and `checkNFC("B\N{COMBINING ACUTE ACCENT}")` will return MAYBE. `"A\N{COMBINING ACUTE ACCENT}"` is not in NFC (its NFC is `"\N{LATIN CAPITAL LETTER A WITH ACUTE}"`), while `"B\N{COMBINING ACUTE ACCENT}"` is in NFC.

If you want to check exactly, compare the string with its NFC/NFKC/FCC.

```
    if ($string eq NFC($string)) {
        # $string is exactly normalized in NFC;
    } else {
        # $string is not normalized in NFC;
```

```
    }

    if ($string eq NFKC($string)) {
        # $string is exactly normalized in NFKC;
    } else {
        # $string is not normalized in NFKC;
    }
```

Character Data

These functions are interface of character data used internally. If you want only to get Unicode normalization forms, you don't need call them yourself.

`$canonical_decomposition = getCanon($code_point)`

If the character is canonically decomposable (including Hangul Syllables), it returns the (full) canonical decomposition as a string. Otherwise it returns `undef`.

Note: According to the Unicode standard, the canonical decomposition of the character that is not canonically decomposable is same as the character itself.

`$compatibility_decomposition = getCompat($code_point)`

If the character is compatibility decomposable (including Hangul Syllables), it returns the (full) compatibility decomposition as a string. Otherwise it returns `undef`.

Note: According to the Unicode standard, the compatibility decomposition of the character that is not compatibility decomposable is same as the character itself.

`$code_point_composite = getComposite($code_point_here, $code_point_next)`

If two characters here and next (as code points) are composable (including Hangul Jamo/Syllables and Composition Exclusions), it returns the code point of the composite.

If they are not composable, it returns `undef`.

`$combining_class = getCombinClass($code_point)`

It returns the combining class (as an integer) of the character.

`$may_be_composed_with_prev_char = isComp2nd($code_point)`

It returns a boolean whether the character of the specified codepoint may be composed with the previous one in a certain composition (including Hangul Compositions, but excluding Composition Exclusions and Non-Starter Decompositions).

`$is_exclusion = isExclusion($code_point)`

It returns a boolean whether the code point is a composition exclusion.

`$is_singleton = isSingleton($code_point)`

It returns a boolean whether the code point is a singleton

`$is_non_starter_decomposition = isNonStDecomp($code_point)`

It returns a boolean whether the code point has Non-Starter Decomposition.

`$is_Full_Composition_Exclusion = isComp_Ex($code_point)`

It returns a boolean of the derived property Comp_Ex (Full_Composition_Exclusion). This property is generated from Composition Exclusions + Singletons + Non-Starter Decompositions.

`$NFD_is_NO = isNFD_NO($code_point)`

It returns a boolean of the derived property NFD_NO (NFD_Quick_Check=No).

`$NFC_is_NO = isNFC_NO($code_point)`

It returns a boolean of the derived property NFC_NO (NFC_Quick_Check=No).

`$NFC_is_MAYBE = isNFC_MAYBE($code_point)`

It returns a boolean of the derived property NFC_MAYBE (NFC_Quick_Check=Maybe).

`$NFKD_is_NO = isNFKD_NO($code_point)`

It returns a boolean of the derived property NFKD_NO (NFKD_Quick_Check=No).

`$NFKC_is_NO = isNFKC_NO($code_point)`

It returns a boolean of the derived property NFKC_NO (NFKC_Quick_Check=No).

`$NFKC_is_MAYBE = isNFKC_MAYBE($code_point)`

It returns a boolean of the derived property NFKC_MAYBE (NFKC_Quick_Check=Maybe).

EXPORT

`NFC`, `NFD`, `NFKC`, `NFKD`: by default.

`normalize` and other some functions: on request.

CAVEATS

Perl's version vs. Unicode version

Since this module refers to perl core's Unicode database in the directory */lib/unicore* (or formerly */lib/unicode*), the Unicode version of normalization implemented by this module depends on your perl's version.

```
perl's version     implemented Unicode version
   5.6.1              3.0.1
   5.7.2              3.1.0
   5.7.3              3.1.1 (normalization is same as 3.1.0)
   5.8.0              3.2.0
 5.8.1-5.8.3          4.0.0
 5.8.4-5.8.6          4.0.1 (normalization is same as 4.0.0)
 5.8.7-5.8.8          4.1.0
   5.10.0             5.0.0
5.8.9, 5.10.1         5.1.0
5.12.0-5.12.3         5.2.0
   5.14.x             6.0.0
   5.16.x             6.1.0
```

Correction of decomposition mapping

In older Unicode versions, a small number of characters (all of which are CJK compatibility ideographs as far as they have been found) may have an erroneous decomposition mapping (see *NormalizationCorrections.txt*). Anyhow, this module will neither refer to *NormalizationCorrections.txt* nor provide any specific version of normalization. Therefore this module running on an older perl with an older Unicode database may use the erroneous decomposition mapping blindly conforming to the Unicode database.

Revised definition of canonical composition

In Unicode 4.1.0, the definition D2 of canonical composition (which affects NFC and NFKC) has been changed (see Public Review Issue #29 and recent UAX #15). This module has used the newer definition since the version 0.07 (Oct 31, 2001). This module will not support the normalization according to the older definition, even if the Unicode version implemented by perl is lower than 4.1.0.

AUTHOR

SADAHIRO Tomoyuki <SADAHIRO@cpan.org>

SEE ALSO

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

Unicode Normalization Forms - UAX #15

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

Composition Exclusion Table

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

Derived Normalization Properties

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

Normalization Corrections

http://www.unicode.org/review/pr-29.html

Public Review Issue #29: Normalization Issue

http://www.unicode.org/notes/tn5/

Canonical Equivalence in Applications - UTN #5