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*).

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

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

Normalization Forms

`$NFD_string = NFD($string)`

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

`$NFC_string = NFC($string)`

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

`$NFKD_string = NFKD($string)`

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

`$NFKC_string = NFKC($string)`

> 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), returns it without modification; otherwise 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)`

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)
```

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)
```

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

Decomposes the specified string and returns the result.

If the second parameter (a boolean) is omitted or false, decomposes it using the Canonical Decomposition Mapping. If true, decomposes it using the Compatibility Decomposition Mapping.

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)
```

Reorders the combining characters and the like in the canonical ordering and returns the result.

E.g., when you have a list of NFD/NFKD strings, you can get the concatenated NFD/NFKD string from them, saying

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

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

Returns the string where composable pairs are composed.

E.g., when you have a NFD/NFKD string, you can get its NFC/NFKC string, saying

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

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:

```
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)
```

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

$result = checkNFC($string)

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

$result = checkNFKD($string)

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

$result = checkNFKC($string)

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

$result = checkFCD($string)

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

$result = checkFCC($string)

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

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)

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.

`$canonical_decomposed = getCanon($codepoint)`

> If the character of the specified codepoint is canonically decomposable (including Hangul Syllables), returns the **completely decomposed** string canonically equivalent to it.
>
> If it is not decomposable, returns `undef`.

`$compatibility_decomposed = getCompat($codepoint)`

> If the character of the specified codepoint is compatibility decomposable (including Hangul Syllables), returns the **completely decomposed** string compatibility equivalent to it.
>
> If it is not decomposable, returns `undef`.

`$codepoint_composite = getComposite($codepoint_here, $codepoint_next)`

> If two characters here and next (as codepoints) are composable (including Hangul Jamo/Syllables and Composition Exclusions), returns the codepoint of the composite.
>
> If they are not composable, returns `undef`.

`$combining_class = getCombinClass($codepoint)`

> Returns the combining class of the character as an integer.

`$is_exclusion = isExclusion($codepoint)`

> Returns a boolean whether the character of the specified codepoint is a composition exclusion.

`$is_singleton = isSingleton($codepoint)`

> Returns a boolean whether the character of the specified codepoint is a singleton.

`$is_non_starter_decomposition = isNonStDecomp($codepoint)`

> Returns a boolean whether the canonical decomposition of the character of the specified codepoint is a Non-Starter Decomposition.

`$may_be_composed_with_prev_char = isComp2nd($codepoint)`

> 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).

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 (same normalized form as that of
3.1.0)
        5.8.0                    3.2.0
```

```
 5.8.1-5.8.3              4.0.0
 5.8.4-5.8.6 (latest)     4.0.1 (same normalized form as that of
4.0.0)
```

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 does not support 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>

Copyright(C) 2001-2005, 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.

SEE ALSO

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

> Unicode Normalization Forms - UAX #15

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