GNU libunistring, version 1.1

updated 4 January 2022

Bruno Haible

Table of Contents

1 Introduction

This library provides functions for manipulating Unicode strings and for manipulating C strings according to the Unicode standard.

It consists of the following parts:

`<unistr.h>`
: elementary string functions

`<uniconv.h>`
: conversion from/to legacy encodings

`<unistdio.h>`
: formatted output to strings

`<uniname.h>`
: character names

`<unictype.h>`
: character classification and properties

`<uniwidth.h>`
: string width when using nonproportional fonts

`<unigbrk.h>`
: grapheme cluster breaks

`<uniwbrk.h>`
: word breaks

`<unilbrk.h>`
: line breaking algorithm

`<uninorm.h>`
: normalization (composition and decomposition)

`<unicase.h>`
: case folding

`<uniregex.h>`
: regular expressions (not yet implemented)

libunistring is for you if your application involves non-trivial text processing, such as upper/lower case conversions, line breaking, operations on words, or more advanced analysis of text. Text provided by the user can, in general, contain characters of all kinds of scripts. The text processing functions provided by this library handle all scripts and all languages.

libunistring is for you if your application already uses the ISO C / POSIX `<ctype.h>`, `<wctype.h>` functions and the text it operates on is provided by the user and can be in any language.

libunistring is also for you if your application uses Unicode strings as internal in-memory representation.

1.1 Unicode

Unicode is a standardized repertoire of characters that contains characters from all scripts of the world, from Latin letters to Chinese ideographs and Babylonian cuneiform glyphs. It also specifies how these characters are to be rendered on a screen or on paper, and how common text processing (word selection, line breaking, uppercasing of page titles etc.) is supposed to behave on Unicode text.

Unicode also specifies three ways of storing sequences of Unicode characters in a computer whose basic unit of data is an 8-bit byte:

UTF-8
: Every character is represented as 1 to 4 bytes.

UTF-16
: Every character is represented as 1 to 2 units of 16 bits.

UTF-32, a.k.a. UCS-4
: Every character is represented as 1 unit of 32 bits.

For encoding Unicode text in a file, UTF-8 is usually used. For encoding Unicode strings in memory for a program, either of the three encoding forms can be reasonably used.

Unicode is widely used on the web. Prior to the use of Unicode, web pages were in many different encodings (ISO-8859-1 for English, French, Spanish, ISO-8859-2 for Polish, ISO-8859-7 for Greek, KOI8-R for Russian, GB2312 or BIG5 for Chinese, ISO-2022-JP-2 or EUC-JP or Shift_JIS for Japanese, and many many others). It was next to impossible to create a document that contained Chinese and Polish text in the same document. Due to the many encodings for Japanese, even the processing of pure Japanese text was error prone.

References:

- The Unicode standard: `https://www.unicode.org/`
- Definition of UTF-8: `https://www.rfc-editor.org/rfc/rfc3629.txt`
- Definition of UTF-16: `https://www.rfc-editor.org/rfc/rfc2781.txt`
- Markus Kuhn's UTF-8 and Unicode FAQ: `https://www.cl.cam.ac.uk/~mgk25/unicode.html`

1.2 Unicode and Internationalization

Internationalization is the process of changing the source code of a program so that it can meet the expectations of users in any culture, if culture specific data (translations, images etc.) are provided.

Use of Unicode is not strictly required for internationalization, but it makes internationalization much easier, because operations that need to look at specific characters (like hyphenation, spell checking, or the automatic conversion of double-quotes to opening and closing double-quote characters) don't need to consider multiple possible encodings of the text.

Use of Unicode also enables multilingualization: the ability of having text in multiple languages present in the same document or even in the same line of text.

But use of Unicode is not everything. Internationalization usually consists of four features:

- Use of Unicode where needed for text processing. This is what this library is for.

- Use of message catalogs for messages shown to the user, This is what GNU gettext is about.
- Use of locale specific conventions for date and time formats, for numeric formatting, or for sorting of text. This can be done adequately with the POSIX APIs and the implementation of locales in the GNU C library.
- In graphical user interfaces, adapting the GUI to the default text direction of the current locale (see right-to-left languages (`https://en.wikipedia.org/wiki/Right-to-left`)).

1.3 Locale encodings

A locale is a set of cultural conventions. According to POSIX, for a program, at any moment, there is one locale being designated as the "current locale". (Actually, POSIX supports also one locale per thread, but this feature is not yet universally implemented and not widely used.) The locale is partitioned into several aspects, called the "categories" of the locale. The main various aspects are:

- The character encoding and the character properties. This is the `LC_CTYPE` category.
- The sorting rules for text. This is the `LC_COLLATE` category.
- The language specific translations of messages. This is the `LC_MESSAGES` category.
- The formatting rules for numbers, such as the decimal separator. This is the `LC_NUMERIC` category.
- The formatting rules for amounts of money. This is the `LC_MONETARY` category.
- The formatting of date and time. This is the `LC_TIME` category.

In particular, the `LC_CTYPE` category of the current locale determines the character encoding. This is the encoding of '`char *`' strings. We also call it the "locale encoding". GNU libunistring has a function, `locale_charset`, that returns a standardized (platform independent) name for this encoding.

All locale encodings used on glibc systems are essentially ASCII compatible: Most graphic ASCII characters have the same representation, as a single byte, in that encoding as in ASCII.

Among the possible locale encodings are UTF-8 and GB18030. Both allow to represent any Unicode character as a sequence of bytes. UTF-8 is used in most of the world, whereas GB18030 is used in the People's Republic of China, because it is backward compatible with the GB2312 encoding that was used in this country earlier.

The legacy locale encodings, ISO-8859-15 (which supplanted ISO-8859-1 in most of Europe), ISO-8859-2, KOI8-R, EUC-JP, etc., are still in use in some places, though.

UTF-16 and UTF-32 are not used as locale encodings, because they are not ASCII compatible.

1.4 Choice of in-memory representation of strings

There are three ways of representing strings in memory of a running program.

- As '`char *`' strings. Such strings are represented in locale encoding. This approach is employed when not much text processing is done by the program. When some Unicode aware processing is to be done, a string is converted to Unicode on the fly and back to locale encoding afterwards.

- As UTF-8 or UTF-16 or UTF-32 strings. This implies that conversion from locale encoding to Unicode is performed on input, and in the opposite direction on output. This approach is employed when the program does a significant amount of text processing, or when the program has multiple threads operating on the same data but in different locales.
- As '`wchar_t *`', a.k.a. "wide strings". This approach is misguided, see Appendix A [The wchar_t mess], page 74.

Of course, a '`char *`' string can, in some cases, be encoded in UTF-8. You will use the data type depending on what you can guarantee about how it's encoded: If a string is encoded in the locale encoding, or if you don't know how it's encoded, use '`char *`'. If, on the other hand, you can *guarantee* that it is UTF-8 encoded, then you can use the UTF-8 string type, `uint8_t *`, for it.

The five types `char *`, `uint8_t *`, `uint16_t *`, `uint32_t *`, and `wchar_t *` are incompatible types at the C level. Therefore, '`gcc -Wall`' will produce a warning if, by mistake, your code contains a mismatch between these types. In the context of using GNU libunistring, even a warning about a mismatch between `char *` and `uint8_t *` is a sign of a bug in your code that you should not try to silence through a cast.

1.5 '`char *`' strings

The classical C strings, with its C library support standardized by ISO C and POSIX, can be used in internationalized programs with some precautions. The problem with this API is that many of the C library functions for strings don't work correctly on strings in locale encodings, leading to bugs that only people in some cultures of the world will experience.

The first problem with the C library API is the support of multibyte locales. According to the locale encoding, in general, every character is represented by one or more bytes (up to 4 bytes in practice — but use `MB_LEN_MAX` instead of the number 4 in the code). When every character is represented by only 1 byte, we speak of an "unibyte locale", otherwise of a "multibyte locale". It is important to realize that the majority of Unix installations nowadays use UTF-8 or GB18030 as locale encoding; therefore, the majority of users are using multibyte locales.

The important fact to remember is:

> *A '`char`' is a byte, not a character.*

As a consequence:

- The `<ctype.h>` API is useless in this context; it does not work in multibyte locales.
- The `strlen` function does not return the number of characters in a string. Nor does it return the number of screen columns occupied by a string after it is output. It merely returns the number of *bytes* occupied by a string.

- Truncating a string, for example, with `strncpy`, can have the effect of truncating it in the middle of a multibyte character. Such a string will, when output, have a garbled character at its end, often represented by a hollow box.
- `strchr` and `strrchr` do not work with multibyte strings if the locale encoding is GB18030 and the character to be searched is a digit.
- `strstr` does not work with multibyte strings if the locale encoding is different from UTF-8.
- `strcspn`, `strpbrk`, `strspn` cannot work correctly in multibyte locales: they assume the second argument is a list of single-byte characters. Even in this simple case, they do not work with multibyte strings if the locale encoding is GB18030 and one of the characters to be searched is a digit.
- `strsep` and `strtok_r` do not work with multibyte strings unless all of the delimiter characters are ASCII characters `<` 0x30.
- The `strcasecmp`, `strncasecmp`, and `strcasestr` functions do not work with multibyte strings.

The workarounds can be found in GNU gnulib `https://www.gnu.org/software/gnulib/`.

- gnulib has modules '`mbchar`', '`mbiter`', '`mbuiter`' that represent multibyte characters and allow to iterate across a multibyte string with the same ease as through a unibyte string.
- gnulib has functions `mbslen` and `mbswidth` that can be used instead of `strlen` when the number of characters or the number of screen columns of a string is requested.
- gnulib has functions `mbschr` and `mbsrrchr` that are like `strchr` and `strrchr`, but work in multibyte locales.
- gnulib has a function `mbsstr`, like `strstr`, but works in multibyte locales.
- gnulib has functions `mbscspn`, `mbspbrk`, `mbsspn` that are like `strcspn`, `strpbrk`, `strspn`, but work in multibyte locales.
- gnulib has functions `mbssep` and `mbstok_r` that are like `strsep` and `strtok_r` but work in multibyte locales.
- gnulib has functions `mbscasecmp`, `mbsncasecmp`, `mbspcasecmp`, and `mbscasestr` that are like `strcasecmp`, `strncasecmp`, and `strcasestr`, but work in multibyte locales. Still, the function `ulc_casecmp` is preferable to these functions; see below.

The second problem with the C library API is that it has some assumptions built-in that are not valid in some languages:

- It assumes that there are only two forms of every character: uppercase and lowercase. This is not true for Croatian, where the character LETTER DZ WITH CARON comes in three forms: LATIN CAPITAL LETTER DZ WITH CARON (DZ), LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON (Dz), LATIN SMALL LETTER DZ WITH CARON (dz).
- It assumes that uppercasing of 1 character leads to 1 character. This is not true for German, where the LATIN SMALL LETTER SHARP S, when uppercased, becomes '`SS`'.

- It assumes that there is 1:1 mapping between uppercase and lowercase forms. This is not true for the Greek sigma: GREEK CAPITAL LETTER SIGMA is the uppercase of both GREEK SMALL LETTER SIGMA and GREEK SMALL LETTER FINAL SIGMA.
- It assumes that the upper/lowercase mappings are position independent. This is not true for the Greek sigma and the Lithuanian i.

The correct way to deal with this problem is

1. to provide functions for titlecasing, as well as for upper- and lowercasing,
2. to view case transformations as functions that operates on strings, rather than on characters.

This is implemented in this library, through the functions declared in `<unicase.h>`, see Chapter 14 [unicase.h], page 62.

1.6 Unicode strings

libunistring supports Unicode strings in three representations:

- UTF-8 strings, through the type '`uint8_t *`'. The units are bytes (`uint8_t`).
- UTF-16 strings, through the type '`uint16_t *`', The units are 16-bit memory words (`uint16_t`).
- UTF-32 strings, through the type '`uint32_t *`'. The units are 32-bit memory words (`uint32_t`).

As with C strings, there are two variants:

- Unicode strings with a terminating NUL character are represented as a pointer to the first unit of the string. There is a unit containing a 0 value at the end. It is considered part of the string for all memory allocation purposes, but is not considered part of the string for all other logical purposes.
- Unicode strings where embedded NUL characters are allowed. These are represented by a pointer to the first unit and the number of units (not bytes!) of the string. In this setting, there is no trailing zero-valued unit used as "end marker".

2 Conventions

This chapter explains conventions valid throughout the libunistring library.

Variables of type `char *` denote C strings in locale encoding. See Section 1.3 [Locale encodings], page 3.

Variables of type `uint8_t *` denote UTF-8 strings. Their units are bytes.

Variables of type `uint16_t *` denote UTF-16 strings, without byte order mark. Their units are 2-byte words.

Variables of type `uint32_t *` denote UTF-32 strings, without byte order mark. Their units are 4-byte words.

Argument pairs `(s, n)` denote a string `s[0..n-1]` with exactly *n* units.

All functions with prefix '`ulc_`' operate on C strings in locale encoding.

All functions with prefix '`u8_`' operate on UTF-8 strings.

All functions with prefix '`u16_`' operate on UTF-16 strings.

All functions with prefix '`u32_`' operate on UTF-32 strings.

For every function with prefix '`u8_`', operating on UTF-8 strings, there is also a corresponding function with prefix '`u16_`', operating on UTF-16 strings, and a corresponding function with prefix '`u32_`', operating on UTF-32 strings. Their description is analogous; in this documentation we describe only the function that operates on UTF-8 strings, for brevity.

A declaration with a variable n denotes the three concrete declarations with $n = 8$, $n = 16$, $n = 32$.

All parameters starting with '`str`' and the parameters of functions starting with `u8_str`/`u16_str`/`u32_str` denote a NUL terminated string.

Error values are always returned through the `errno` variable, usually with a return value that indicates the presence of an error (NULL for functions that return an pointer, or -1 for functions that return an `int`).

Functions returning a string result take a `(resultbuf, lengthp)` argument pair. If *resultbuf* is not NULL and the result fits into `*lengthp` units, it is put in *resultbuf*, and *resultbuf* is returned. Otherwise, a freshly allocated string is returned. In both cases, `*lengthp` is set to the length (number of units) of the returned string. In case of error, NULL is returned and `errno` is set.

3 Elementary types <unitypes.h>

The include file <unitypes.h> provides the following basic types.

uint8_t [Type]
uint16_t [Type]
uint32_t [Type]

These are the storage units of UTF-8/16/32 strings, respectively. The definitions are taken from <stdint.h>, on platforms where this include file is present.

ucs4_t [Type]

This type represents a single Unicode character, outside of an UTF-32 string.

The types ucs4_t and uint32_t happen to be identical. They differ in use and intent, however:

- Use uint32_t * to designate an UTF-32 string. Use ucs4_t to designate a single Unicode character, outside of an UTF-32 string.
- Conversions functions that take an UTF-32 string as input will usually perform a range-check on the uint32_t values. Whereas functions that are declared to take ucs4_t arguments will not perform such a range-check.

4 Elementary Unicode string functions <unistr.h>

This include file declares elementary functions for Unicode strings. It is essentially the equivalent of what `<string.h>` is for C strings.

4.1 Elementary string checks

The following function is available to verify the integrity of a Unicode string.

`const uint8_t * u8_check` (*const uint8_t* **s*, *size_t* *n*) [Function]
`const uint16_t * u16_check` (*const uint16_t* **s*, *size_t* *n*) [Function]
`const uint32_t * u32_check` (*const uint32_t* **s*, *size_t* *n*) [Function]

This function checks whether a Unicode string is well-formed. It returns NULL if valid, or a pointer to the first invalid unit otherwise.

4.2 Elementary string conversions

The following functions perform conversions between the different forms of Unicode strings.

`uint16_t * u8_to_u16` (*const uint8_t* **s*, *size_t* *n*, *uint16_t* **resultbuf*, *size_t* **lengthp*) [Function]

Converts an UTF-8 string to an UTF-16 string.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

`uint32_t * u8_to_u32` (*const uint8_t* **s*, *size_t* *n*, *uint32_t* **resultbuf*, *size_t* **lengthp*) [Function]

Converts an UTF-8 string to an UTF-32 string.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

`uint8_t * u16_to_u8` (*const uint16_t* **s*, *size_t* *n*, *uint8_t* **resultbuf*, *size_t* **lengthp*) [Function]

Converts an UTF-16 string to an UTF-8 string.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

`uint32_t * u16_to_u32` (*const uint16_t* **s*, *size_t* *n*, *uint32_t* **resultbuf*, *size_t* **lengthp*) [Function]

Converts an UTF-16 string to an UTF-32 string.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

`uint8_t * u32_to_u8` (*const uint32_t* **s*, *size_t* *n*, *uint8_t* **resultbuf*, *size_t* **lengthp*) [Function]

Converts an UTF-32 string to an UTF-8 string.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

uint16_t * u32_to_u16 (*const uint32_t *s, size_t n, uint16_t *resultbuf, size_t *lengthp*) [Function]

Converts an UTF-32 string to an UTF-16 string.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

4.3 Elementary string functions

4.3.1 Iterating over a Unicode string

The following functions inspect and return details about the first character in a Unicode string.

int u8_mblen (*const uint8_t *s, size_t n*) [Function]
int u16_mblen (*const uint16_t *s, size_t n*) [Function]
int u32_mblen (*const uint32_t *s, size_t n*) [Function]

Returns the length (number of units) of the first character in *s*, which is no longer than *n*. Returns 0 if it is the NUL character. Returns -1 upon failure.

This function is similar to `mblen`, except that it operates on a Unicode string and that *s* must not be NULL.

int u8_mbtouc (*ucs4_t *puc, const uint8_t *s, size_t n*) [Function]
int u16_mbtouc (*ucs4_t *puc, const uint16_t *s, size_t n*) [Function]
int u32_mbtouc (*ucs4_t *puc, const uint32_t *s, size_t n*) [Function]

Returns the length (number of units) of the first character in *s*, putting its `ucs4_t` representation in `*puc`. Upon failure, `*puc` is set to `0xfffd`, and an appropriate number of units is returned.

The number of available units, *n*, must be > 0.

This function fails if an invalid sequence of units is encountered at the beginning of *s*, or if additional units (after the *n* provided units) would be needed to form a character.

This function is similar to `mbtowc`, except that it operates on a Unicode string, *puc* and *s* must not be NULL, *n* must be > 0, and the NUL character is not treated specially.

int u8_mbtouc_unsafe (*ucs4_t *puc, const uint8_t *s, size_t n*) [Function]
int u16_mbtouc_unsafe (*ucs4_t *puc, const uint16_t *s, size_t n*) [Function]
int u32_mbtouc_unsafe (*ucs4_t *puc, const uint32_t *s, size_t n*) [Function]

This function is identical to `u8_mbtouc/u16_mbtouc/u32_mbtouc`. Earlier versions of this function performed fewer range-checks on the sequence of units.

int u8_mbtoucr (*ucs4_t *puc, const uint8_t *s, size_t n*) [Function]
int u16_mbtoucr (*ucs4_t *puc, const uint16_t *s, size_t n*) [Function]
int u32_mbtoucr (*ucs4_t *puc, const uint32_t *s, size_t n*) [Function]

Returns the length (number of units) of the first character in *s*, putting its `ucs4_t` representation in `*puc`. Upon failure, `*puc` is set to `0xfffd`, and -1 is returned for an invalid sequence of units, -2 is returned for an incomplete sequence of units.

The number of available units, *n*, must be > 0.

This function is similar to `u8_mbtouc`, except that the return value gives more details about the failure, similar to `mbrtowc`.

4.3.2 Creating Unicode strings one character at a time

The following function stores a Unicode character as a Unicode string in memory.

`int u8_uctomb` (*uint8_t *s*, *ucs4_t uc*, *ptrdiff_t n*) [Function]
`int u16_uctomb` (*uint16_t *s*, *ucs4_t uc*, *ptrdiff_t n*) [Function]
`int u32_uctomb` (*uint32_t *s*, *ucs4_t uc*, *ptrdiff_t n*) [Function]

Puts the multibyte character represented by *uc* in *s*, returning its length. Returns -1 upon failure, -2 if the number of available units, *n*, is too small. The latter case cannot occur if *n* >= 6/2/1, respectively.

This function is similar to `wctomb`, except that it operates on a Unicode strings, *s* must not be NULL, and the argument *n* must be specified.

4.3.3 Copying Unicode strings

The following functions copy Unicode strings in memory.

`uint8_t * u8_cpy` (*uint8_t *dest*, *const uint8_t *src*, *size_t n*) [Function]
`uint16_t * u16_cpy` (*uint16_t *dest*, *const uint16_t *src*, *size_t n*) [Function]
`uint32_t * u32_cpy` (*uint32_t *dest*, *const uint32_t *src*, *size_t n*) [Function]

Copies *n* units from *src* to *dest*.

This function is similar to `memcpy`, except that it operates on Unicode strings.

`uint8_t * u8_move` (*uint8_t *dest*, *const uint8_t *src*, *size_t n*) [Function]
`uint16_t * u16_move` (*uint16_t *dest*, *const uint16_t *src*, *size_t n*) [Function]
`uint32_t * u32_move` (*uint32_t *dest*, *const uint32_t *src*, *size_t n*) [Function]

Copies *n* units from *src* to *dest*, guaranteeing correct behavior for overlapping memory areas.

This function is similar to `memmove`, except that it operates on Unicode strings.

The following function fills a Unicode string.

`uint8_t * u8_set` (*uint8_t *s*, *ucs4_t uc*, *size_t n*) [Function]
`uint16_t * u16_set` (*uint16_t *s*, *ucs4_t uc*, *size_t n*) [Function]
`uint32_t * u32_set` (*uint32_t *s*, *ucs4_t uc*, *size_t n*) [Function]

Sets the first *n* characters of *s* to *uc*. *uc* should be a character that occupies only 1 unit.

This function is similar to `memset`, except that it operates on Unicode strings.

4.3.4 Comparing Unicode strings

The following function compares two Unicode strings of the same length.

`int u8_cmp` (*const uint8_t *s1*, *const uint8_t *s2*, *size_t n*) [Function]
`int u16_cmp` (*const uint16_t *s1*, *const uint16_t *s2*, *size_t n*) [Function]
`int u32_cmp` (*const uint32_t *s1*, *const uint32_t *s2*, *size_t n*) [Function]

Compares *s1* and *s2*, each of length *n*, lexicographically. Returns a negative value if *s1* compares smaller than *s2*, a positive value if *s1* compares larger than *s2*, or 0 if they compare equal.

This function is similar to `memcmp`, except that it operates on Unicode strings.

The following function compares two Unicode strings of possibly different lengths.

`int u8_cmp2` (*const uint8_t* **s1*, *size_t* *n1*, *const uint8_t* **s2*, *size_t* *n2*) [Function]
`int u16_cmp2` (*const uint16_t* **s1*, *size_t* *n1*, *const uint16_t* **s2*, *size_t* *n2*) [Function]
`int u32_cmp2` (*const uint32_t* **s1*, *size_t* *n1*, *const uint32_t* **s2*, *size_t* *n2*) [Function]

Compares *s1* and *s2*, lexicographically. Returns a negative value if *s1* compares smaller than *s2*, a positive value if *s1* compares larger than *s2*, or 0 if they compare equal.

This function is similar to the gnulib function `memcmp2`, except that it operates on Unicode strings.

4.3.5 Searching for a character in a Unicode string

The following function searches for a given Unicode character.

`uint8_t * u8_chr` (*const uint8_t* **s*, *size_t* *n*, *ucs4_t* *uc*) [Function]
`uint16_t * u16_chr` (*const uint16_t* **s*, *size_t* *n*, *ucs4_t* *uc*) [Function]
`uint32_t * u32_chr` (*const uint32_t* **s*, *size_t* *n*, *ucs4_t* *uc*) [Function]

Searches the string at *s* for *uc*. Returns a pointer to the first occurrence of *uc* in *s*, or NULL if *uc* does not occur in *s*.

This function is similar to `memchr`, except that it operates on Unicode strings.

4.3.6 Counting the characters in a Unicode string

The following function counts the number of Unicode characters.

`size_t u8_mbsnlen` (*const uint8_t* **s*, *size_t* *n*) [Function]
`size_t u16_mbsnlen` (*const uint16_t* **s*, *size_t* *n*) [Function]
`size_t u32_mbsnlen` (*const uint32_t* **s*, *size_t* *n*) [Function]

Counts and returns the number of Unicode characters in the *n* units from *s*.

This function is similar to the gnulib function `mbsnlen`, except that it operates on Unicode strings.

4.4 Elementary string functions with memory allocation

The following function copies a Unicode string.

`uint8_t * u8_cpy_alloc` (*const uint8_t* **s*, *size_t* *n*) [Function]
`uint16_t * u16_cpy_alloc` (*const uint16_t* **s*, *size_t* *n*) [Function]
`uint32_t * u32_cpy_alloc` (*const uint32_t* **s*, *size_t* *n*) [Function]

Makes a freshly allocated copy of *s*, of length *n*.

4.5 Elementary string functions on NUL terminated strings

4.5.1 Iterating over a NUL terminated Unicode string

The following functions inspect and return details about the first character in a Unicode string.

`int u8_strmblen` (*const uint8_t *s*) [Function]
`int u16_strmblen` (*const uint16_t *s*) [Function]
`int u32_strmblen` (*const uint32_t *s*) [Function]

Returns the length (number of units) of the first character in *s*. Returns 0 if it is the NUL character. Returns -1 upon failure.

`int u8_strmbtouc` (*ucs4_t *puc, const uint8_t *s*) [Function]
`int u16_strmbtouc` (*ucs4_t *puc, const uint16_t *s*) [Function]
`int u32_strmbtouc` (*ucs4_t *puc, const uint32_t *s*) [Function]

Returns the length (number of units) of the first character in *s*, putting its `ucs4_t` representation in `*puc`. Returns 0 if it is the NUL character. Returns -1 upon failure.

`const uint8_t * u8_next` (*ucs4_t *puc, const uint8_t *s*) [Function]
`const uint16_t * u16_next` (*ucs4_t *puc, const uint16_t *s*) [Function]
`const uint32_t * u32_next` (*ucs4_t *puc, const uint32_t *s*) [Function]

Forward iteration step. Advances the pointer past the next character, or returns NULL if the end of the string has been reached. Puts the character's `ucs4_t` representation in `*puc`.

The following function inspects and returns details about the previous character in a Unicode string.

`const uint8_t * u8_prev` (*ucs4_t *puc, const uint8_t *s, const uint8_t *start*) [Function]
`const uint16_t * u16_prev` (*ucs4_t *puc, const uint16_t *s, const uint16_t *start*) [Function]
`const uint32_t * u32_prev` (*ucs4_t *puc, const uint32_t *s, const uint32_t *start*) [Function]

Backward iteration step. Advances the pointer to point to the previous character (the one that ends at `s`), or returns NULL if the beginning of the string (specified by `start`) had been reached. Puts the character's `ucs4_t` representation in `*puc`. Note that this function works only on well-formed Unicode strings.

4.5.2 Length of a NUL terminated Unicode string

The following functions determine the length of a Unicode string.

`size_t u8_strlen` (*const uint8_t *s*) [Function]
`size_t u16_strlen` (*const uint16_t *s*) [Function]
`size_t u32_strlen` (*const uint32_t *s*) [Function]

Returns the number of units in *s*.

This function is similar to `strlen` and `wcslen`, except that it operates on Unicode strings.

size_t **u8_strnlen** (*const uint8_t *s, size_t maxlen*) [Function]
size_t **u16_strnlen** (*const uint16_t *s, size_t maxlen*) [Function]
size_t **u32_strnlen** (*const uint32_t *s, size_t maxlen*) [Function]

Returns the number of units in *s*, but at most *maxlen*.

This function is similar to `strnlen` and `wcsnlen`, except that it operates on Unicode strings.

4.5.3 Copying a NUL terminated Unicode string

The following functions copy portions of Unicode strings in memory.

uint8_t * **u8_strcpy** (*uint8_t *dest, const uint8_t *src*) [Function]
uint16_t * **u16_strcpy** (*uint16_t *dest, const uint16_t *src*) [Function]
uint32_t * **u32_strcpy** (*uint32_t *dest, const uint32_t *src*) [Function]

Copies *src* to *dest*.

This function is similar to `strcpy` and `wcscpy`, except that it operates on Unicode strings.

uint8_t * **u8_stpcpy** (*uint8_t *dest, const uint8_t *src*) [Function]
uint16_t * **u16_stpcpy** (*uint16_t *dest, const uint16_t *src*) [Function]
uint32_t * **u32_stpcpy** (*uint32_t *dest, const uint32_t *src*) [Function]

Copies *src* to *dest*, returning the address of the terminating NUL in *dest*.

This function is similar to `stpcpy`, except that it operates on Unicode strings.

uint8_t * **u8_strncpy** (*uint8_t *dest, const uint8_t *src, size_t n*) [Function]
uint16_t * **u16_strncpy** (*uint16_t *dest, const uint16_t *src, size_t n*) [Function]
uint32_t * **u32_strncpy** (*uint32_t *dest, const uint32_t *src, size_t n*) [Function]

Copies no more than *n* units of *src* to *dest*.

This function is similar to `strncpy` and `wcsncpy`, except that it operates on Unicode strings.

uint8_t * **u8_stpncpy** (*uint8_t *dest, const uint8_t *src, size_t n*) [Function]
uint16_t * **u16_stpncpy** (*uint16_t *dest, const uint16_t *src, size_t n*) [Function]
uint32_t * **u32_stpncpy** (*uint32_t *dest, const uint32_t *src, size_t n*) [Function]

Copies no more than *n* units of *src* to *dest*. Returns a pointer past the last non-NUL unit written into *dest*. In other words, if the units written into *dest* include a NUL, the return value is the address of the first such NUL unit, otherwise it is `dest + n`.

This function is similar to `stpncpy`, except that it operates on Unicode strings.

uint8_t * **u8_strcat** (*uint8_t *dest, const uint8_t *src*) [Function]
uint16_t * **u16_strcat** (*uint16_t *dest, const uint16_t *src*) [Function]
uint32_t * **u32_strcat** (*uint32_t *dest, const uint32_t *src*) [Function]

Appends *src* onto *dest*.

This function is similar to `strcat` and `wcscat`, except that it operates on Unicode strings.

uint8_t * u8_strncat (*uint8_t *dest, const uint8_t *src, size_t n*) [Function]
uint16_t * u16_strncat (*uint16_t *dest, const uint16_t *src, size_t n*) [Function]
uint32_t * u32_strncat (*uint32_t *dest, const uint32_t *src, size_t n*) [Function]

Appends no more than *n* units of *src* onto *dest*.

This function is similar to `strncat` and `wcsncat`, except that it operates on Unicode strings.

4.5.4 Comparing NUL terminated Unicode strings

The following functions compare two Unicode strings.

int u8_strcmp (*const uint8_t *s1, const uint8_t *s2*) [Function]
int u16_strcmp (*const uint16_t *s1, const uint16_t *s2*) [Function]
int u32_strcmp (*const uint32_t *s1, const uint32_t *s2*) [Function]

Compares *s1* and *s2*, lexicographically. Returns a negative value if *s1* compares smaller than *s2*, a positive value if *s1* compares larger than *s2*, or 0 if they compare equal.

This function is similar to `strcmp` and `wcscmp`, except that it operates on Unicode strings.

int u8_strcoll (*const uint8_t *s1, const uint8_t *s2*) [Function]
int u16_strcoll (*const uint16_t *s1, const uint16_t *s2*) [Function]
int u32_strcoll (*const uint32_t *s1, const uint32_t *s2*) [Function]

Compares *s1* and *s2* using the collation rules of the current locale. Returns -1 if $s1 < s2$, 0 if $s1 = s2$, 1 if $s1 > s2$. Upon failure, sets `errno` and returns any value.

This function is similar to `strcoll` and `wcscoll`, except that it operates on Unicode strings.

Note that this function may consider different canonical normalizations of the same string as having a large distance. It is therefore better to use the function `u8_normcoll` instead of this one; see Chapter 13 [uninorm.h], page 57.

int u8_strncmp (*const uint8_t *s1, const uint8_t *s2, size_t n*) [Function]
int u16_strncmp (*const uint16_t *s1, const uint16_t *s2, size_t n*) [Function]
int u32_strncmp (*const uint32_t *s1, const uint32_t *s2, size_t n*) [Function]

Compares no more than *n* units of *s1* and *s2*.

This function is similar to `strncmp` and `wcsncmp`, except that it operates on Unicode strings.

4.5.5 Duplicating a NUL terminated Unicode string

The following function allocates a duplicate of a Unicode string.

uint8_t * u8_strdup (*const uint8_t *s*) [Function]
uint16_t * u16_strdup (*const uint16_t *s*) [Function]
uint32_t * u32_strdup (*const uint32_t *s*) [Function]

Duplicates *s*, returning an identical malloc'd string.

This function is similar to `strdup` and `wcsdup`, except that it operates on Unicode strings.

4.5.6 Searching for a character in a NUL terminated Unicode string

The following functions search for a given Unicode character.

uint8_t * u8_strchr (*const uint8_t *str, ucs4_t uc*) [Function]
uint16_t * u16_strchr (*const uint16_t *str, ucs4_t uc*) [Function]
uint32_t * u32_strchr (*const uint32_t *str, ucs4_t uc*) [Function]

Finds the first occurrence of *uc* in *str*.

This function is similar to `strchr` and `wcschr`, except that it operates on Unicode strings.

uint8_t * u8_strrchr (*const uint8_t *str, ucs4_t uc*) [Function]
uint16_t * u16_strrchr (*const uint16_t *str, ucs4_t uc*) [Function]
uint32_t * u32_strrchr (*const uint32_t *str, ucs4_t uc*) [Function]

Finds the last occurrence of *uc* in *str*.

This function is similar to `strrchr` and `wcsrchr`, except that it operates on Unicode strings.

The following functions search for the first occurrence of some Unicode character in or outside a given set of Unicode characters.

size_t u8_strcspn (*const uint8_t *str, const uint8_t *reject*) [Function]
size_t u16_strcspn (*const uint16_t *str, const uint16_t *reject*) [Function]
size_t u32_strcspn (*const uint32_t *str, const uint32_t *reject*) [Function]

Returns the length of the initial segment of *str* which consists entirely of Unicode characters not in *reject*.

This function is similar to `strcspn` and `wcscspn`, except that it operates on Unicode strings.

size_t u8_strspn (*const uint8_t *str, const uint8_t *accept*) [Function]
size_t u16_strspn (*const uint16_t *str, const uint16_t *accept*) [Function]
size_t u32_strspn (*const uint32_t *str, const uint32_t *accept*) [Function]

Returns the length of the initial segment of *str* which consists entirely of Unicode characters in *accept*.

This function is similar to `strspn` and `wcsspn`, except that it operates on Unicode strings.

uint8_t * u8_strpbrk (*const uint8_t *str, const uint8_t *accept*) [Function]
uint16_t * u16_strpbrk (*const uint16_t *str, const uint16_t *accept*) [Function]
uint32_t * u32_strpbrk (*const uint32_t *str, const uint32_t *accept*) [Function]

Finds the first occurrence in *str* of any character in *accept*.

This function is similar to `strpbrk` and `wcspbrk`, except that it operates on Unicode strings.

4.5.7 Searching for a substring in a NUL terminated Unicode string

The following functions search whether a given Unicode string is a substring of another Unicode string.

`uint8_t * u8_strstr` (*const uint8_t *`haystack`*, const uint8_t *`needle`*) [Function]
`uint16_t * u16_strstr` (*const uint16_t *`haystack`*, const uint16_t *`needle`*) [Function]
`uint32_t * u32_strstr` (*const uint32_t *`haystack`*, const uint32_t *`needle`*) [Function]

Finds the first occurrence of *needle* in *haystack*.

This function is similar to `strstr` and `wcsstr`, except that it operates on Unicode strings.

`bool u8_startswith` (*const uint8_t *`str`*, const uint8_t *`prefix`*) [Function]
`bool u16_startswith` (*const uint16_t *`str`*, const uint16_t *`prefix`*) [Function]
`bool u32_startswith` (*const uint32_t *`str`*, const uint32_t *`prefix`*) [Function]

Tests whether *str* starts with *prefix*.

`bool u8_endswith` (*const uint8_t *`str`*, const uint8_t *`suffix`*) [Function]
`bool u16_endswith` (*const uint16_t *`str`*, const uint16_t *`suffix`*) [Function]
`bool u32_endswith` (*const uint32_t *`str`*, const uint32_t *`suffix`*) [Function]

Tests whether *str* ends with *suffix*.

4.5.8 Tokenizing a NUL terminated Unicode string

The following function does one step in tokenizing a Unicode string.

`uint8_t * u8_strtok` (*uint8_t *`str`*, const uint8_t *`delim`*, uint8_t **`ptr`*) [Function]
`uint16_t * u16_strtok` (*uint16_t *`str`*, const uint16_t *`delim`*, uint16_t **`ptr`*) [Function]
`uint32_t * u32_strtok` (*uint32_t *`str`*, const uint32_t *`delim`*, uint32_t **`ptr`*) [Function]

Divides *str* into tokens separated by characters in *delim*.

This function is similar to `strtok_r` and `wcstok`, except that it operates on Unicode strings. Its interface is actually more similar to `wcstok` than to `strtok`.

5 Conversions between Unicode and encodings <uniconv.h>

This include file declares functions for converting between Unicode strings and `char *` strings in locale encoding or in other specified encodings.

The following function returns the locale encoding.

`const char * locale_charset ()` [Function]

Determines the current locale's character encoding, and canonicalizes it into one of the canonical names listed in `localcharset.h`. If the canonical name cannot be determined, the result is a non-canonical name.

The result must not be freed; it is statically allocated.

The result of this function can be used as an argument to the `iconv_open` function in GNU libc, in GNU libiconv, or in the gnulib provided wrapper around the native `iconv_open` function. It may not work as an argument to the native `iconv_open` function directly.

The handling of unconvertible characters during the conversions can be parametrized through the following enumeration type:

`enum iconv_ilseq_handler` [Type]

This type specifies how unconvertible characters in the input are handled.

`enum iconv_ilseq_handler iconveh_error` [Constant]

This handler causes the function to return with `errno` set to `EILSEQ`.

`enum iconv_ilseq_handler iconveh_question_mark` [Constant]

This handler produces one question mark '`?`' per unconvertible character.

`enum iconv_ilseq_handler iconveh_question_replacement_character` [Constant]

This handler produces one U+FFFD per unconvertible character if that fits in the target encoding, otherwise one question mark '`?`' per unconvertible character.

`enum iconv_ilseq_handler iconveh_escape_sequence` [Constant]

This handler produces an escape sequence `\uxxxx` or `\Uxxxxxxxx` for each unconvertible character.

The following functions convert between strings in a specified encoding and Unicode strings.

`uint8_t * u8_conv_from_encoding` (*const char* ****fromcode***, *enum iconv_ilseq_handler* ***handler***, *const char* ****src***, *size_t* ***srclen***, *size_t* ****offsets***, *uint8_t* ****resultbuf***, *size_t* ****lengthp***) [Function]

`uint16_t * u16_conv_from_encoding` (*const char* ****fromcode***, *enum iconv_ilseq_handler* ***handler***, *const char* ****src***, *size_t* ***srclen***, *size_t* ****offsets***, *uint16_t* ****resultbuf***, *size_t* ****lengthp***) [Function]

`uint32_t * u32_conv_from_encoding` (*const char* ****fromcode***, *enum iconv_ilseq_handler* ***handler***, *const char* ****src***, *size_t* ***srclen***, *size_t* ****offsets***, *uint32_t* ****resultbuf***, *size_t* ****lengthp***) [Function]

Converts an entire string, possibly including NUL bytes, from one encoding to UTF-8 encoding.

Converts a memory region given in encoding *fromcode*. *fromcode* is as for the `iconv_open` function.

The input is in the memory region between *src* (inclusive) and *src* + *srclen* (exclusive).

If *offsets* is not NULL, it should point to an array of *srclen* integers; this array is filled with offsets into the result, i.e. the character starting at *src*[i] corresponds to the character starting at *result*[*offsets*[i]], and other offsets are set to `(size_t)(-1)`.

resultbuf and **lengthp* should be a scratch buffer and its size, or *resultbuf* can be NULL.

May erase the contents of the memory at *resultbuf*.

If successful: The resulting Unicode string (non-NULL) is returned and its length stored in **lengthp*. The resulting string is *resultbuf* if no dynamic memory allocation was necessary, or a freshly allocated memory block otherwise.

In case of error: NULL is returned and `errno` is set. Particular `errno` values: `EINVAL`, `EILSEQ`, `ENOMEM`.

`char * u8_conv_to_encoding` (*const char *tocode*, *enum iconv_ilseq_handler handler*, *const uint8_t *src*, *size_t srclen*, *size_t *offsets*, *char *resultbuf*, *size_t *lengthp*) [Function]

`char * u16_conv_to_encoding` (*const char *tocode*, *enum iconv_ilseq_handler handler*, *const uint16_t *src*, *size_t srclen*, *size_t *offsets*, *char *resultbuf*, *size_t *lengthp*) [Function]

`char * u32_conv_to_encoding` (*const char *tocode*, *enum iconv_ilseq_handler handler*, *const uint32_t *src*, *size_t srclen*, *size_t *offsets*, *char *resultbuf*, *size_t *lengthp*) [Function]

Converts an entire Unicode string, possibly including NUL units, from UTF-8 encoding to a given encoding.

Converts a memory region to encoding *tocode*. *tocode* is as for the `iconv_open` function.

The input is in the memory region between *src* (inclusive) and *src* + *srclen* (exclusive).

If *offsets* is not NULL, it should point to an array of *srclen* integers; this array is filled with offsets into the result, i.e. the character starting at *src*[i] corresponds to the character starting at *result*[*offsets*[i]], and other offsets are set to `(size_t)(-1)`.

resultbuf and **lengthp* should be a scratch buffer and its size, or *resultbuf* can be NULL.

May erase the contents of the memory at *resultbuf*.

If successful: The resulting Unicode string (non-NULL) is returned and its length stored in **lengthp*. The resulting string is *resultbuf* if no dynamic memory allocation was necessary, or a freshly allocated memory block otherwise.

In case of error: NULL is returned and `errno` is set. Particular `errno` values: `EINVAL`, `EILSEQ`, `ENOMEM`.

The following functions convert between NUL terminated strings in a specified encoding and NUL terminated Unicode strings.

`uint8_t * u8_strconv_from_encoding` (*const char *`string`, const char *`fromcode`, enum iconv_ilseq_handler `handler`*) [Function]
`uint16_t * u16_strconv_from_encoding` (*const char *`string`, const char *`fromcode`, enum iconv_ilseq_handler `handler`*) [Function]
`uint32_t * u32_strconv_from_encoding` (*const char *`string`, const char *`fromcode`, enum iconv_ilseq_handler `handler`*) [Function]

Converts a NUL terminated string from a given encoding.

The result is `malloc` allocated, or NULL (with *errno* set) in case of error.

Particular `errno` values: `EILSEQ`, `ENOMEM`.

`char * u8_strconv_to_encoding` (*const uint8_t *`string`, const char *`tocode`, enum iconv_ilseq_handler `handler`*) [Function]
`char * u16_strconv_to_encoding` (*const uint16_t *`string`, const char *`tocode`, enum iconv_ilseq_handler `handler`*) [Function]
`char * u32_strconv_to_encoding` (*const uint32_t *`string`, const char *`tocode`, enum iconv_ilseq_handler `handler`*) [Function]

Converts a NUL terminated string to a given encoding.

The result is `malloc` allocated, or NULL (with `errno` set) in case of error.

Particular `errno` values: `EILSEQ`, `ENOMEM`.

The following functions are shorthands that convert between NUL terminated strings in locale encoding and NUL terminated Unicode strings.

`uint8_t * u8_strconv_from_locale` (*const char *`string`*) [Function]
`uint16_t * u16_strconv_from_locale` (*const char *`string`*) [Function]
`uint32_t * u32_strconv_from_locale` (*const char *`string`*) [Function]

Converts a NUL terminated string from the locale encoding.

The result is `malloc` allocated, or NULL (with `errno` set) in case of error.

Particular `errno` values: `ENOMEM`.

`char * u8_strconv_to_locale` (*const uint8_t *`string`*) [Function]
`char * u16_strconv_to_locale` (*const uint16_t *`string`*) [Function]
`char * u32_strconv_to_locale` (*const uint32_t *`string`*) [Function]

Converts a NUL terminated string to the locale encoding.

The result is `malloc` allocated, or NULL (with `errno` set) in case of error.

Particular `errno` values: `ENOMEM`.

6 Output with Unicode strings <unistdio.h>

This include file declares functions for doing formatted output with Unicode strings. It defines a set of functions similar to `fprintf` and `sprintf`, which are declared in `<stdio.h>`.

These functions work like the `printf` function family. In the format string:

- The format directive '`U`' takes an UTF-8 string (`const uint8_t *`).
- The format directive '`lU`' takes an UTF-16 string (`const uint16_t *`).
- The format directive '`llU`' takes an UTF-32 string (`const uint32_t *`).

A function name with an infix '`v`' indicates that a `va_list` is passed instead of multiple arguments.

The functions `*sprintf` have a *buf* argument that is assumed to be large enough. (*DANGEROUS! Overflowing the buffer will crash the program.*)

The functions `*snprintf` have a *buf* argument that is assumed to be *size* units large. (*DANGEROUS! The resulting string might be truncated in the middle of a multibyte character.*)

The functions `*asprintf` have a *resultp* argument. The result will be freshly allocated and stored in `*resultp`.

The functions `*asnprintf` have a (*resultbuf*, *lengthp*) argument pair. If *resultbuf* is not NULL and the result fits into `*lengthp` units, it is put in *resultbuf*, and *resultbuf* is returned. Otherwise, a freshly allocated string is returned. In both cases, `*lengthp` is set to the length (number of units) of the returned string. In case of error, NULL is returned and `errno` is set.

The following functions take an ASCII format string and return a result that is a `char *` string in locale encoding.

`int ulc_sprintf` (*char* *`buf`, *const char* *`format`, ...) [Function]

`int ulc_snprintf` (*char* *`buf`, *size_t size*, *const char* *`format`, ...) [Function]

`int ulc_asprintf` (*char* **`resultp`, *const char* *`format`, ...) [Function]

`char * ulc_asnprintf` (*char* *`resultbuf`, *size_t* *`lengthp`, *const char* *`format`, ...) [Function]

`int ulc_vsprintf` (*char* *`buf`, *const char* *`format`, *va_list* `ap`) [Function]

`int ulc_vsnprintf` (*char* *`buf`, *size_t size*, *const char* *`format`, *va_list* `ap`) [Function]

`int ulc_vasprintf` (*char* **`resultp`, *const char* *`format`, *va_list* `ap`) [Function]

`char * ulc_vasnprintf` (*char* *`resultbuf`, *size_t* *`lengthp`, *const char* *`format`, *va_list* `ap`) [Function]

The following functions take an ASCII format string and return a result in UTF-8 format.

`int u8_sprintf` (*uint8_t* *`buf`, *const char* *`format`, ...) [Function]

`int u8_snprintf` (*uint8_t* *`buf`, *size_t* `size`, *const char* *`format`, ...) [Function]

`int u8_asprintf` (*uint8_t* **`resultp`, *const char* *`format`, ...) [Function] `uint8_t * u8_asnprintf` (*uint8_t *resultbuf*, *size_t *lengthp*, *const char *format*, ...) [Function]

`int u8_vsprintf` (*uint8_t *buf*, *const char *format*, *va_list ap*) [Function]

`int u8_vsnprintf` (*uint8_t *buf*, *size_t size*, *const char *format*, *va_list ap*) [Function]

`int u8_vasprintf` (*uint8_t **resultp*, *const char *format*, *va_list ap*) [Function]

`uint8_t * u8_vasnprintf` (*uint8_t *resultbuf*, *size_t *lengthp*, *const char *format*, *va_list ap*) [Function]

The following functions take an UTF-8 format string and return a result in UTF-8 format.

`int u8_u8_sprintf` (*uint8_t *buf*, *const uint8_t *format*, ...) [Function]

`int u8_u8_snprintf` (*uint8_t *buf*, *size_t size*, *const uint8_t *format*, ...) [Function]

`int u8_u8_asprintf` (*uint8_t **resultp*, *const uint8_t *format*, ...) [Function]

`uint8_t * u8_u8_asnprintf` (*uint8_t *resultbuf*, *size_t *lengthp*, *const uint8_t *format*, ...) [Function]

`int u8_u8_vsprintf` (*uint8_t *buf*, *const uint8_t *format*, *va_list ap*) [Function]

`int u8_u8_vsnprintf` (*uint8_t *buf*, *size_t size*, *const uint8_t *format*, *va_list ap*) [Function]

`int u8_u8_vasprintf` (*uint8_t **resultp*, *const uint8_t *format*, *va_list ap*) [Function]

`uint8_t * u8_u8_vasnprintf` (*uint8_t *resultbuf*, *size_t *lengthp*, *const uint8_t *format*, *va_list ap*) [Function]

The following functions take an ASCII format string and return a result in UTF-16 format.

`int u16_sprintf` (*uint16_t *buf*, *const char *format*, ...) [Function]

`int u16_snprintf` (*uint16_t *buf*, *size_t size*, *const char *format*, ...) [Function]

`int u16_asprintf` (*uint16_t **resultp*, *const char *format*, ...) [Function]

`uint16_t * u16_asnprintf` (*uint16_t *resultbuf*, *size_t *lengthp*, *const char *format*, ...) [Function]

`int u16_vsprintf` (*uint16_t *buf*, *const char *format*, *va_list ap*) [Function]

`int u16_vsnprintf` (*uint16_t *buf*, *size_t size*, *const char *format*, *va_list ap*) [Function] int u16_vasprintf (*uint16_t* ***resultp*, *const char* **format*, *va_list* ap) [Function]

uint16_t * u16_vasnprintf (*uint16_t* **resultbuf*, *size_t* **lengthp*, *const char* **format*, *va_list* ap) [Function]

The following functions take an UTF-16 format string and return a result in UTF-16 format.

int u16_u16_sprintf (*uint16_t* **buf*, *const uint16_t* **format*, ...) [Function]

int u16_u16_snprintf (*uint16_t* **buf*, *size_t* size, *const uint16_t* **format*, ...) [Function]

int u16_u16_asprintf (*uint16_t* ***resultp*, *const uint16_t* **format*, ...) [Function]

uint16_t * u16_u16_asnprintf (*uint16_t* **resultbuf*, *size_t* **lengthp*, *const uint16_t* **format*, ...) [Function]

int u16_u16_vsprintf (*uint16_t* **buf*, *const uint16_t* **format*, *va_list* ap) [Function]

int u16_u16_vsnprintf (*uint16_t* **buf*, *size_t* size, *const uint16_t* **format*, *va_list* ap) [Function]

int u16_u16_vasprintf (*uint16_t* ***resultp*, *const uint16_t* **format*, *va_list* ap) [Function]

uint16_t * u16_u16_vasnprintf (*uint16_t* **resultbuf*, *size_t* **lengthp*, *const uint16_t* **format*, *va_list* ap) [Function]

The following functions take an ASCII format string and return a result in UTF-32 format.

int u32_sprintf (*uint32_t* **buf*, *const char* **format*, ...) [Function]

int u32_snprintf (*uint32_t* **buf*, *size_t* size, *const char* **format*, ...) [Function]

int u32_asprintf (*uint32_t* ***resultp*, *const char* **format*, ...) [Function]

uint32_t * u32_asnprintf (*uint32_t* **resultbuf*, *size_t* **lengthp*, *const char* **format*, ...) [Function]

int u32_vsprintf (*uint32_t* **buf*, *const char* **format*, *va_list ap*) [Function]

int u32_vsnprintf (*uint32_t* **buf*, *size_t* size, *const char* **format*, *va_list* ap) [Function]

int u32_vasprintf (*uint32_t* ***resultp*, *const char* **format*, *va_list* ap) [Function]

uint32_t * u32_vasnprintf (*uint32_t* **resultbuf*, *size_t* **lengthp*, *const char* **format*, *va_list* ap) [Function]

The following functions take an UTF-32 format string and return a result in UTF-32 format.

int u32_u32_sprintf (*uint32_t* *buf, *const uint32_t* *format, ...) [Function]

int u32_u32_snprintf (*uint32_t* *buf, *size_t* size, *const uint32_t* *format, ...) [Function]

int u32_u32_asprintf (*uint32_t* **resultp, *const uint32_t* *format, ...) [Function]

uint32_t * u32_u32_asnprintf (*uint32_t* *resultbuf, *size_t* *lengthp, *const uint32_t* *format, ...) [Function]

int u32_u32_vsprintf (*uint32_t* *buf, *const uint32_t* *format, *va_list* ap) [Function]

int u32_u32_vsnprintf (*uint32_t* *buf, *size_t* size, *const uint32_t* *format, *va_list* ap) [Function]

int u32_u32_vasprintf (*uint32_t* **resultp, *const uint32_t* *format, *va_list* ap) [Function]

uint32_t * u32_u32_vasnprintf (*uint32_t* *resultbuf, *size_t* *lengthp, *const uint32_t* *format, *va_list* ap) [Function]

The following functions take an ASCII format string and produce output in locale encoding to a FILE stream.

int ulc_fprintf (*FILE* *stream, *const char* *format, ...) [Function]

int ulc_vfprintf (*FILE* *stream, *const char* *format, *va_list* ap) [Function]

7 Names of Unicode characters <uniname.h>

This include file implements the association between a Unicode character and its name.

The name of a Unicode character allows to distinguish it from other, similar looking characters. For example, the character 'x' has the name "LATIN SMALL LETTER X" and is therefore different from the character named "MULTIPLICATION SIGN".

`unsigned int UNINAME_MAX` [Macro]

This macro expands to a constant that is the required size of buffer for a Unicode character name.

`char * unicode_character_name` (*ucs4_t* `uc`, *char* *`*buf`*) [Function]

Looks up the name of a Unicode character, in uppercase ASCII. *buf* must point to a buffer, at least `UNINAME_MAX` bytes in size. Returns the filled *buf*, or NULL if the character does not have a name.

`ucs4_t unicode_name_character` (*const char* *`*name`*) [Function]

Looks up the Unicode character with a given name, in upper- or lowercase ASCII. *NAME* can also be an alias name of a character. Returns the character if found, or `UNINAME_INVALID` if not found.

`ucs4_t UNINAME_INVALID` [Macro]

This macro expands to a constant that is a special return value of the `unicode_name_character` function.

8 Unicode character classification and properties <unictype.h>

This include file declares functions that classify Unicode characters and that test whether Unicode characters have specific properties.

The classification assigns a "general category" to every Unicode character. This is similar to the classification provided by ISO C in <wctype.h>.

Properties are the data that guides various text processing algorithms in the presence of specific Unicode characters.

8.1 General category

Every Unicode character or code point has a *general category* assigned to it. This classification is important for most algorithms that work on Unicode text.

The GNU libunistring library provides two kinds of API for working with general categories. The object oriented API uses a variable to denote every predefined general category value or combinations thereof. The low-level API uses a bit mask instead. The advantage of the object oriented API is that if only a few predefined general category values are used, the data tables are relatively small. When you combine general category values (using `uc_general_category_or`, `uc_general_category_and`, or `uc_general_category_and_not`), or when you use the low level bit masks, a big table is used thats holds the complete general category information for all Unicode characters.

8.1.1 The object oriented API for general category

`uc_general_category_t` [Type]

This data type denotes a general category value. It is an immediate type that can be copied by simple assignment, without involving memory allocation. It is not an array type.

The following are the predefined general category value. Additional general categories may be added in the future.

The `UC_CATEGORY_*` constants reflect the systematic general category values assigned by the Unicode Consortium. Whereas the other `UC_*` macros are aliases, for use when readable code is preferred.

`uc_general_category_t UC_CATEGORY_L` [Constant]
`uc_general_category_t UC_LETTER` [Macro]

This represents the general category "Letter".

`uc_general_category_t UC_CATEGORY_LC` [Constant]
`uc_general_category_t UC_CASED_LETTER` [Macro]

`uc_general_category_t UC_CATEGORY_Lu` [Constant]
`uc_general_category_t UC_UPPERCASE_LETTER` [Macro]

This represents the general category "Letter, uppercase".

`uc_general_category_t UC_CATEGORY_Ll` [Constant]
`uc_general_category_t UC_LOWERCASE_LETTER` [Macro]

This represents the general category "Letter, lowercase".

uc_general_category_t UC_CATEGORY_Lt [Constant]
uc_general_category_t UC_TITLECASE_LETTER [Macro]
This represents the general category "Letter, titlecase".

uc_general_category_t UC_CATEGORY_Lm [Constant]
uc_general_category_t UC_MODIFIER_LETTER [Macro]
This represents the general category "Letter, modifier".

uc_general_category_t UC_CATEGORY_Lo [Constant]
uc_general_category_t UC_OTHER_LETTER [Macro]
This represents the general category "Letter, other".

uc_general_category_t UC_CATEGORY_M [Constant]
uc_general_category_t UC_MARK [Macro]
This represents the general category "Marker".

uc_general_category_t UC_CATEGORY_Mn [Constant]
uc_general_category_t UC_NON_SPACING_MARK [Macro]
This represents the general category "Marker, nonspacing".

uc_general_category_t UC_CATEGORY_Mc [Constant]
uc_general_category_t UC_COMBINING_SPACING_MARK [Macro]
This represents the general category "Marker, spacing combining".

uc_general_category_t UC_CATEGORY_Me [Constant]
uc_general_category_t UC_ENCLOSING_MARK [Macro]
This represents the general category "Marker, enclosing".

uc_general_category_t UC_CATEGORY_N [Constant]
uc_general_category_t UC_NUMBER [Macro]
This represents the general category "Number".

uc_general_category_t UC_CATEGORY_Nd [Constant]
uc_general_category_t UC_DECIMAL_DIGIT_NUMBER [Macro]
This represents the general category "Number, decimal digit".

uc_general_category_t UC_CATEGORY_Nl [Constant]
uc_general_category_t UC_LETTER_NUMBER [Macro]
This represents the general category "Number, letter".

uc_general_category_t UC_CATEGORY_No [Constant]
uc_general_category_t UC_OTHER_NUMBER [Macro]
This represents the general category "Number, other".

uc_general_category_t UC_CATEGORY_P [Constant]
uc_general_category_t UC_PUNCTUATION [Macro]
This represents the general category "Punctuation".

uc_general_category_t UC_CATEGORY_Pc [Constant]
uc_general_category_t UC_CONNECTOR_PUNCTUATION [Macro]
This represents the general category "Punctuation, connector".

uc_general_category_t UC_CATEGORY_Pd [Constant]
uc_general_category_t UC_DASH_PUNCTUATION [Macro]

This represents the general category "Punctuation, dash".

uc_general_category_t UC_CATEGORY_Ps [Constant]
uc_general_category_t UC_OPEN_PUNCTUATION [Macro]

This represents the general category "Punctuation, open", a.k.a. "start punctuation".

uc_general_category_t UC_CATEGORY_Pe [Constant]
uc_general_category_t UC_CLOSE_PUNCTUATION [Macro]

This represents the general category "Punctuation, close", a.k.a. "end punctuation".

uc_general_category_t UC_CATEGORY_Pi [Constant]
uc_general_category_t UC_INITIAL_QUOTE_PUNCTUATION [Macro]

This represents the general category "Punctuation, initial quote".

uc_general_category_t UC_CATEGORY_Pf [Constant]
uc_general_category_t UC_FINAL_QUOTE_PUNCTUATION [Macro]

This represents the general category "Punctuation, final quote".

uc_general_category_t UC_CATEGORY_Po [Constant]
uc_general_category_t UC_OTHER_PUNCTUATION [Macro]

This represents the general category "Punctuation, other".

uc_general_category_t UC_CATEGORY_S [Constant]
uc_general_category_t UC_SYMBOL [Macro]

This represents the general category "Symbol".

uc_general_category_t UC_CATEGORY_Sm [Constant]
uc_general_category_t UC_MATH_SYMBOL [Macro]

This represents the general category "Symbol, math".

uc_general_category_t UC_CATEGORY_Sc [Constant]
uc_general_category_t UC_CURRENCY_SYMBOL [Macro]

This represents the general category "Symbol, currency".

uc_general_category_t UC_CATEGORY_Sk [Constant]
uc_general_category_t UC_MODIFIER_SYMBOL [Macro]

This represents the general category "Symbol, modifier".

uc_general_category_t UC_CATEGORY_So [Constant]
uc_general_category_t UC_OTHER_SYMBOL [Macro]

This represents the general category "Symbol, other".

uc_general_category_t UC_CATEGORY_Z [Constant]
uc_general_category_t UC_SEPARATOR [Macro]

This represents the general category "Separator".

uc_general_category_t UC_CATEGORY_Zs [Constant]
uc_general_category_t UC_SPACE_SEPARATOR [Macro]

This represents the general category "Separator, space".

`uc_general_category_t UC_CATEGORY_Zl` [Constant]
`uc_general_category_t UC_LINE_SEPARATOR` [Macro]
This represents the general category "Separator, line".

`uc_general_category_t UC_CATEGORY_Zp` [Constant]
`uc_general_category_t UC_PARAGRAPH_SEPARATOR` [Macro]
This represents the general category "Separator, paragraph".

`uc_general_category_t UC_CATEGORY_C` [Constant]
`uc_general_category_t UC_OTHER` [Macro]
This represents the general category "Other".

`uc_general_category_t UC_CATEGORY_Cc` [Constant]
`uc_general_category_t UC_CONTROL` [Macro]
This represents the general category "Other, control".

`uc_general_category_t UC_CATEGORY_Cf` [Constant]
`uc_general_category_t UC_FORMAT` [Macro]
This represents the general category "Other, format".

`uc_general_category_t UC_CATEGORY_Cs` [Constant]
`uc_general_category_t UC_SURROGATE` [Macro]
This represents the general category "Other, surrogate". All code points in this category are invalid characters.

`uc_general_category_t UC_CATEGORY_Co` [Constant]
`uc_general_category_t UC_PRIVATE_USE` [Macro]
This represents the general category "Other, private use".

`uc_general_category_t UC_CATEGORY_Cn` [Constant]
`uc_general_category_t UC_UNASSIGNED` [Macro]
This represents the general category "Other, not assigned". Some code points in this category are invalid characters.

The following functions combine general categories, like in a boolean algebra, except that there is no '`not`' operation.

`uc_general_category_t uc_general_category_or` [Function]
(*uc_general_category_t* `category1`, *uc_general_category_t* `category2`)
Returns the union of two general categories. This corresponds to the unions of the two sets of characters.

`uc_general_category_t uc_general_category_and` [Function]
(*uc_general_category_t* `category1`, *uc_general_category_t* `category2`)
Returns the intersection of two general categories as bit masks. This *does not* correspond to the intersection of the two sets of characters.

`uc_general_category_t uc_general_category_and_not` [Function]
(*uc_general_category_t* `category1`, *uc_general_category_t* `category2`)
Returns the intersection of a general category with the complement of a second general category, as bit masks. This *does not* correspond to the intersection with complement, when viewing the categories as sets of characters.

The following functions associate general categories with their name.

`const char * uc_general_category_name` [Function]
(*uc_general_category_t* `category`)

Returns the name of a general category, more precisely, the abbreviated name. Returns NULL if the general category corresponds to a bit mask that does not have a name.

`const char * uc_general_category_long_name` [Function]
(*uc_general_category_t* `category`)

Returns the long name of a general category. Returns NULL if the general category corresponds to a bit mask that does not have a name.

`uc_general_category_t uc_general_category_byname` [Function]
(*const char* *`category_name`)

Returns the general category given by name, e.g. `"Lu"`, or by long name, e.g. `"Uppercase Letter"`. This lookup ignores spaces, underscores, or hyphens as word separators and is case-insignificant.

The following functions view general categories as sets of Unicode characters.

`uc_general_category_t uc_general_category` (*ucs4_t* `uc`) [Function]

Returns the general category of a Unicode character.

This function uses a big table.

`bool uc_is_general_category` (*ucs4_t* `uc`, [Function]
uc_general_category_t `category`)

Tests whether a Unicode character belongs to a given category. The *category* argument can be a predefined general category or the combination of several predefined general categories.

8.1.2 The bit mask API for general category

The following are the predefined general category value as bit masks. Additional general categories may be added in the future.

`uint32_t UC_CATEGORY_MASK_L` [Macro]
`uint32_t UC_CATEGORY_MASK_LC` [Macro]
`uint32_t UC_CATEGORY_MASK_Lu` [Macro]
`uint32_t UC_CATEGORY_MASK_Ll` [Macro]
`uint32_t UC_CATEGORY_MASK_Lt` [Macro]
`uint32_t UC_CATEGORY_MASK_Lm` [Macro]
`uint32_t UC_CATEGORY_MASK_Lo` [Macro]
`uint32_t UC_CATEGORY_MASK_M` [Macro]
`uint32_t UC_CATEGORY_MASK_Mn` [Macro]
`uint32_t UC_CATEGORY_MASK_Mc` [Macro]
`uint32_t UC_CATEGORY_MASK_Me` [Macro]
`uint32_t UC_CATEGORY_MASK_N` [Macro]
`uint32_t UC_CATEGORY_MASK_Nd` [Macro]
`uint32_t UC_CATEGORY_MASK_Nl` [Macro]

uint32_t UC_CATEGORY_MASK_No [Macro]
uint32_t UC_CATEGORY_MASK_P [Macro]
uint32_t UC_CATEGORY_MASK_Pc [Macro]
uint32_t UC_CATEGORY_MASK_Pd [Macro]
uint32_t UC_CATEGORY_MASK_Ps [Macro]
uint32_t UC_CATEGORY_MASK_Pe [Macro]
uint32_t UC_CATEGORY_MASK_Pi [Macro]
uint32_t UC_CATEGORY_MASK_Pf [Macro]
uint32_t UC_CATEGORY_MASK_Po [Macro]
uint32_t UC_CATEGORY_MASK_S [Macro]
uint32_t UC_CATEGORY_MASK_Sm [Macro]
uint32_t UC_CATEGORY_MASK_Sc [Macro]
uint32_t UC_CATEGORY_MASK_Sk [Macro]
uint32_t UC_CATEGORY_MASK_So [Macro]
uint32_t UC_CATEGORY_MASK_Z [Macro]
uint32_t UC_CATEGORY_MASK_Zs [Macro]
uint32_t UC_CATEGORY_MASK_Zl [Macro]
uint32_t UC_CATEGORY_MASK_Zp [Macro]
uint32_t UC_CATEGORY_MASK_C [Macro]
uint32_t UC_CATEGORY_MASK_Cc [Macro]
uint32_t UC_CATEGORY_MASK_Cf [Macro]
uint32_t UC_CATEGORY_MASK_Cs [Macro]
uint32_t UC_CATEGORY_MASK_Co [Macro]
uint32_t UC_CATEGORY_MASK_Cn [Macro]

The following function views general categories as sets of Unicode characters.

bool uc_is_general_category_withtable (*ucs4_t* uc, *uint32_t* bitmask) [Function]

Tests whether a Unicode character belongs to a given category. The *bitmask* argument can be a predefined general category bitmask or the combination of several predefined general category bitmasks.

This function uses a big table comprising all general categories.

8.2 Canonical combining class

Every Unicode character or code point has a *canonical combining class* assigned to it.

What is the meaning of the canonical combining class? Essentially, it indicates the priority with which a combining character is attached to its base character. The characters for which the canonical combining class is 0 are the base characters, and the characters for which it is greater than 0 are the combining characters. Combining characters are rendered near/attached/around their base character, and combining characters with small combining classes are attached "first" or "closer" to the base character.

The canonical combining class of a character is a number in the range 0..255. The possible values are described in the Unicode Character Database https://www.unicode.org/Public/UNIDATA/UCD.html. The list here is not definitive; more values can be added in future versions.

`int UC_CCC_NR` [Constant]
The canonical combining class value for "Not Reordered" characters. The value is 0.

`int UC_CCC_OV` [Constant]
The canonical combining class value for "Overlay" characters.

`int UC_CCC_NK` [Constant]
The canonical combining class value for "Nukta" characters.

`int UC_CCC_KV` [Constant]
The canonical combining class value for "Kana Voicing" characters.

`int UC_CCC_VR` [Constant]
The canonical combining class value for "Virama" characters.

`int UC_CCC_ATBL` [Constant]
The canonical combining class value for "Attached Below Left" characters.

`int UC_CCC_ATB` [Constant]
The canonical combining class value for "Attached Below" characters.

`int UC_CCC_ATA` [Constant]
The canonical combining class value for "Attached Above" characters.

`int UC_CCC_ATAR` [Constant]
The canonical combining class value for "Attached Above Right" characters.

`int UC_CCC_BL` [Constant]
The canonical combining class value for "Below Left" characters.

`int UC_CCC_B` [Constant]
The canonical combining class value for "Below" characters.

`int UC_CCC_BR` [Constant]
The canonical combining class value for "Below Right" characters.

`int UC_CCC_L` [Constant]
The canonical combining class value for "Left" characters.

`int UC_CCC_R` [Constant]
The canonical combining class value for "Right" characters.

`int UC_CCC_AL` [Constant]
The canonical combining class value for "Above Left" characters.

`int UC_CCC_A` [Constant]
The canonical combining class value for "Above" characters.

`int UC_CCC_AR` [Constant]
The canonical combining class value for "Above Right" characters.

`int UC_CCC_DB` [Constant]
The canonical combining class value for "Double Below" characters.

int UC_CCC_DA [Constant]
The canonical combining class value for "Double Above" characters.

int UC_CCC_IS [Constant]
The canonical combining class value for "Iota Subscript" characters.

The following functions associate canonical combining classes with their name.

const char * uc_combining_class_name (*int ccc*) [Function]
Returns the name of a canonical combining class, more precisely, the abbreviated name. Returns NULL if the canonical combining class is a numeric value without a name.

const char * uc_combining_class_long_name (*int ccc*) [Function]
Returns the long name of a canonical combining class. Returns NULL if the canonical combining class is a numeric value without a name.

int uc_combining_class_byname (*const char *ccc_name*) [Function]
Returns the canonical combining class given by name, e.g. "BL", or by long name, e.g. "Below Left". This lookup ignores spaces, underscores, or hyphens as word separators and is case-insignificant.

The following function looks up the canonical combining class of a character.

int uc_combining_class (*ucs4_t uc*) [Function]
Returns the canonical combining class of a Unicode character.

8.3 Bidi class

Every Unicode character or code point has a *bidi class* assigned to it. Before Unicode 4.0, this concept was known as *bidirectional category*.

The bidi class guides the bidirectional algorithm (https://www.unicode.org/reports/tr9/). The possible values are the following.

int UC_BIDI_L [Constant]
The bidi class for 'Left-to-Right'" characters.

int UC_BIDI_LRE [Constant]
The bidi class for "Left-to-Right Embedding" characters.

int UC_BIDI_LRO [Constant]
The bidi class for "Left-to-Right Override" characters.

int UC_BIDI_R [Constant]
The bidi class for "Right-to-Left" characters.

int UC_BIDI_AL [Constant]
The bidi class for "Right-to-Left Arabic" characters.

int UC_BIDI_RLE [Constant]
The bidi class for "Right-to-Left Embedding" characters.

`int UC_BIDI_RLO` [Constant]
: The bidi class for "Right-to-Left Override" characters.

`int UC_BIDI_PDF` [Constant]
: The bidi class for "Pop Directional Format" characters.

`int UC_BIDI_EN` [Constant]
: The bidi class for "European Number" characters.

`int UC_BIDI_ES` [Constant]
: The bidi class for "European Number Separator" characters.

`int UC_BIDI_ET` [Constant]
: The bidi class for "European Number Terminator" characters.

`int UC_BIDI_AN` [Constant]
: The bidi class for "Arabic Number" characters.

`int UC_BIDI_CS` [Constant]
: The bidi class for "Common Number Separator" characters.

`int UC_BIDI_NSM` [Constant]
: The bidi class for "Non-Spacing Mark" characters.

`int UC_BIDI_BN` [Constant]
: The bidi class for "Boundary Neutral" characters.

`int UC_BIDI_B` [Constant]
: The bidi class for "Paragraph Separator" characters.

`int UC_BIDI_S` [Constant]
: The bidi class for "Segment Separator" characters.

`int UC_BIDI_WS` [Constant]
: The bidi class for "Whitespace" characters.

`int UC_BIDI_ON` [Constant]
: The bidi class for "Other Neutral" characters.

`int UC_BIDI_LRI` [Constant]
: The bidi class for "Left-to-Right Isolate" characters.

`int UC_BIDI_RLI` [Constant]
: The bidi class for "Right-to-Left Isolate" characters.

`int UC_BIDI_FSI` [Constant]
: The bidi class for "First Strong Isolate" characters.

`int UC_BIDI_PDI` [Constant]
: The bidi class for "Pop Directional Isolate" characters.

The following functions implement the association between a bidirectional category and its name.

`const char * uc_bidi_class_name` (*int* `bidi_class`) [Function]
`const char * uc_bidi_category_name` (*int* `category`) [Function]

Returns the name of a bidi class, more precisely, the abbreviated name.

`const char * uc_bidi_class_long_name` (*int* `bidi_class`) [Function]

Returns the long name of a bidi class.

`int uc_bidi_class_byname` (*const char* *`bidi_class_name`) [Function]
`int uc_bidi_category_byname` (*const char* *`category_name`) [Function]

Returns the bidi class given by name, e.g. `"LRE"`, or by long name, e.g. `"Left-to-Right Embedding"`. This lookup ignores spaces, underscores, or hyphens as word separators and is case-insignificant.

The following functions view bidirectional categories as sets of Unicode characters.

`int uc_bidi_class` (*ucs4_t* `uc`) [Function]
`int uc_bidi_category` (*ucs4_t* `uc`) [Function]

Returns the bidi class of a Unicode character.

`bool uc_is_bidi_class` (*ucs4_t* `uc`, *int* `bidi_class`) [Function]
`bool uc_is_bidi_category` (*ucs4_t* `uc`, *int* `category`) [Function]

Tests whether a Unicode character belongs to a given bidi class.

8.4 Decimal digit value

Decimal digits (like the digits from ‘`0`’ to ‘`9`’) exist in many scripts. The following function converts a decimal digit character to its numerical value.

`int uc_decimal_value` (*ucs4_t* `uc`) [Function]

Returns the decimal digit value of a Unicode character. The return value is an integer in the range 0..9, or -1 for characters that do not represent a decimal digit.

8.5 Digit value

Digit characters are like decimal digit characters, possibly in special forms, like as superscript, subscript, or circled. The following function converts a digit character to its numerical value.

`int uc_digit_value` (*ucs4_t* `uc`) [Function]

Returns the digit value of a Unicode character. The return value is an integer in the range 0..9, or -1 for characters that do not represent a digit.

8.6 Numeric value

There are also characters that represent numbers without a digit system, like the Roman numerals, and fractional numbers, like 1/4 or 3/4.

The following type represents the numeric value of a Unicode character.

`uc_fraction_t` [Type]

This is a structure type with the following fields:

```
int numerator;
```

```
int denominator;
```

An integer *n* is represented by `numerator = n`, `denominator = 1`.

The following function converts a number character to its numerical value.

`uc_fraction_t uc_numeric_value` (*ucs4_t uc*) [Function]

Returns the numeric value of a Unicode character. The return value is a fraction, or the pseudo-fraction `{ 0, 0 }` for characters that do not represent a number.

8.7 Mirrored character

Character mirroring is used to associate the closing parenthesis character to the opening parenthesis character, the closing brace character with the opening brace character, and so on.

The following function looks up the mirrored character of a Unicode character.

`bool uc_mirror_char` (*ucs4_t uc, ucs4_t *puc*) [Function]

Stores the mirrored character of a Unicode character *uc* in **puc* and returns `true`, if it exists. Otherwise it stores *uc* unmodified in **puc* and returns `false`.

8.8 Arabic shaping

When Arabic characters are rendered, after bidi reordering has taken place, the shape of the glyphs are modified so that many adjacent glyphs are joined. Two character properties describe how this "Arabic shaping" takes place: the joining type and the joining group.

8.8.1 Joining type of Arabic characters

The joining type of a character describes on which of the left and right neighbour characters the character's shape depends, and which of the two neighbour characters are rendered depending on this character.

The joining type has the following possible values:

`int UC_JOINING_TYPE_U` [Constant]

"Non joining": Characters of this joining type prohibit joining.

`int UC_JOINING_TYPE_T` [Constant]

"Transparent": Characters of this joining type are skipped when considering joining.

`int UC_JOINING_TYPE_C` [Constant]

"Join causing": Characters of this joining type cause their neighbour characters to change their shapes but don't change their own shape.

`int UC_JOINING_TYPE_L` [Constant]

"Left joining": Characters of this joining type have two shapes, isolated and initial. Such characters currently don't exist.

`int UC_JOINING_TYPE_R` [Constant]

"Right joining": Characters of this joining type have two shapes, isolated and final.

`int UC_JOINING_TYPE_D` [Constant]
"Dual joining": Characters of this joining type have four shapes, initial, medial, final, and isolated.

The following functions implement the association between a joining type and its name.

`const char * uc_joining_type_name (`*int* `joining_type)` [Function]
Returns the name of a joining type.

`const char * uc_joining_type_long_name (`*int* `joining_type)` [Function]
Returns the long name of a joining type.

`int uc_joining_type_byname (`*const char* `*joining_type_name)` [Function]
Returns the joining type given by name, e.g. `"D"`, or by long name, e.g. `"Dual Joining`. This lookup ignores spaces, underscores, or hyphens as word separators and is case-insignificant.

The following function gives the joining type of every Unicode character.

`int uc_joining_type (`*ucs4_t* `uc)` [Function]
Returns the joining type of a Unicode character.

8.8.2 Joining group of Arabic characters

The joining group of a character describes how the character's shape is modified in the four contexts of dual-joining characters or in the two contexts of right-joining characters.

The joining group has the following possible values:

`int UC_JOINING_GROUP_NONE` [Constant]
`int UC_JOINING_GROUP_AIN` [Constant]
`int UC_JOINING_GROUP_ALAPH` [Constant]
`int UC_JOINING_GROUP_ALEF` [Constant]
`int UC_JOINING_GROUP_BEH` [Constant]
`int UC_JOINING_GROUP_BETH` [Constant]
`int UC_JOINING_GROUP_BURUSHASKI_YEH_BARREE` [Constant]
`int UC_JOINING_GROUP_DAL` [Constant]
`int UC_JOINING_GROUP_DALATH_RISH` [Constant]
`int UC_JOINING_GROUP_E` [Constant]
`int UC_JOINING_GROUP_FARSI_YEH` [Constant]
`int UC_JOINING_GROUP_FE` [Constant]
`int UC_JOINING_GROUP_FEH` [Constant]
`int UC_JOINING_GROUP_FINAL_SEMKATH` [Constant]
`int UC_JOINING_GROUP_GAF` [Constant]
`int UC_JOINING_GROUP_GAMAL` [Constant]
`int UC_JOINING_GROUP_HAH` [Constant]
`int UC_JOINING_GROUP_HE` [Constant]
`int UC_JOINING_GROUP_HEH` [Constant]
`int UC_JOINING_GROUP_HEH_GOAL` [Constant]
`int UC_JOINING_GROUP_HETH` [Constant]
`int UC_JOINING_GROUP_KAF` [Constant]

int UC_JOINING_GROUP_KAPH [Constant]
int UC_JOINING_GROUP_KHAPH [Constant]
int UC_JOINING_GROUP_KNOTTED_HEH [Constant]
int UC_JOINING_GROUP_LAM [Constant]
int UC_JOINING_GROUP_LAMADH [Constant]
int UC_JOINING_GROUP_MEEM [Constant]
int UC_JOINING_GROUP_MIM [Constant]
int UC_JOINING_GROUP_NOON [Constant]
int UC_JOINING_GROUP_NUN [Constant]
int UC_JOINING_GROUP_NYA [Constant]
int UC_JOINING_GROUP_PE [Constant]
int UC_JOINING_GROUP_QAF [Constant]
int UC_JOINING_GROUP_QAPH [Constant]
int UC_JOINING_GROUP_REH [Constant]
int UC_JOINING_GROUP_REVERSED_PE [Constant]
int UC_JOINING_GROUP_SAD [Constant]
int UC_JOINING_GROUP_SADHE [Constant]
int UC_JOINING_GROUP_SEEN [Constant]
int UC_JOINING_GROUP_SEMKATH [Constant]
int UC_JOINING_GROUP_SHIN [Constant]
int UC_JOINING_GROUP_SWASH_KAF [Constant]
int UC_JOINING_GROUP_SYRIAC_WAW [Constant]
int UC_JOINING_GROUP_TAH [Constant]
int UC_JOINING_GROUP_TAW [Constant]
int UC_JOINING_GROUP_TEH_MARBUTA [Constant]
int UC_JOINING_GROUP_TEH_MARBUTA_GOAL [Constant]
int UC_JOINING_GROUP_TETH [Constant]
int UC_JOINING_GROUP_WAW [Constant]
int UC_JOINING_GROUP_YEH [Constant]
int UC_JOINING_GROUP_YEH_BARREE [Constant]
int UC_JOINING_GROUP_YEH_WITH_TAIL [Constant]
int UC_JOINING_GROUP_YUDH [Constant]
int UC_JOINING_GROUP_YUDH_HE [Constant]
int UC_JOINING_GROUP_ZAIN [Constant]
int UC_JOINING_GROUP_ZHAIN [Constant]
int UC_JOINING_GROUP_ROHINGYA_YEH [Constant]
int UC_JOINING_GROUP_STRAIGHT_WAW [Constant]
int UC_JOINING_GROUP_MANICHAEAN_ALEPH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_BETH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_GIMEL [Constant]
int UC_JOINING_GROUP_MANICHAEAN_DALETH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_WAW [Constant]
int UC_JOINING_GROUP_MANICHAEAN_ZAYIN [Constant]
int UC_JOINING_GROUP_MANICHAEAN_HETH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_TETH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_YODH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_KAPH [Constant]

int UC_JOINING_GROUP_MANICHAEAN_LAMEDH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_DHAMEDH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_THAMEDH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_MEM [Constant]
int UC_JOINING_GROUP_MANICHAEAN_NUN [Constant]
int UC_JOINING_GROUP_MANICHAEAN_SAMEKH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_AYIN [Constant]
int UC_JOINING_GROUP_MANICHAEAN_PE [Constant]
int UC_JOINING_GROUP_MANICHAEAN_SADHE [Constant]
int UC_JOINING_GROUP_MANICHAEAN_QOPH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_RESH [Constant]
int UC_JOINING_GROUP_MANICHAEAN_TAW [Constant]
int UC_JOINING_GROUP_MANICHAEAN_ONE [Constant]
int UC_JOINING_GROUP_MANICHAEAN_FIVE [Constant]
int UC_JOINING_GROUP_MANICHAEAN_TEN [Constant]
int UC_JOINING_GROUP_MANICHAEAN_TWENTY [Constant]
int UC_JOINING_GROUP_MANICHAEAN_HUNDRED [Constant]
int UC_JOINING_GROUP_AFRICAN_FEH [Constant]
int UC_JOINING_GROUP_AFRICAN_QAF [Constant]
int UC_JOINING_GROUP_AFRICAN_NOON [Constant]
int UC_JOINING_GROUP_MALAYALAM_NGA [Constant]
int UC_JOINING_GROUP_MALAYALAM_JA [Constant]
int UC_JOINING_GROUP_MALAYALAM_NYA [Constant]
int UC_JOINING_GROUP_MALAYALAM_TTA [Constant]
int UC_JOINING_GROUP_MALAYALAM_NNA [Constant]
int UC_JOINING_GROUP_MALAYALAM_NNNA [Constant]
int UC_JOINING_GROUP_MALAYALAM_BHA [Constant]
int UC_JOINING_GROUP_MALAYALAM_RA [Constant]
int UC_JOINING_GROUP_MALAYALAM_LLA [Constant]
int UC_JOINING_GROUP_MALAYALAM_LLLA [Constant]
int UC_JOINING_GROUP_MALAYALAM_SSA [Constant]
int UC_JOINING_GROUP_HANIFI_ROHINGYA_PA [Constant]
int UC_JOINING_GROUP_HANIFI_ROHINGYA_KINNA_YA [Constant]
int UC_JOINING_GROUP_THIN_YEH [Constant]
int UC_JOINING_GROUP_VERTICAL_TAIL [Constant]

The following functions implement the association between a joining group and its name.

const char * uc_joining_group_name (*int joining_group*) [Function]

Returns the name of a joining group.

int uc_joining_group_byname (*const char *joining_group_name*) [Function]

Returns the joining group given by name, e.g. "Teh_Marbuta". This lookup ignores spaces, underscores, or hyphens as word separators and is case-insignificant.

The following function gives the joining group of every Unicode character.

int uc_joining_group (*ucs4_t uc*) [Function]

Returns the joining group of a Unicode character.

8.9 Properties

This section defines boolean properties of Unicode characters. This means, a character either has the given property or does not have it. In other words, the property can be viewed as a subset of the set of Unicode characters.

The GNU libunistring library provides two kinds of API for working with properties. The object oriented API uses a type `uc_property_t` to designate a property. In the function-based API, which is a bit more low level, a property is merely a function.

8.9.1 Properties as objects – the object oriented API

The following type designates a property on Unicode characters.

`uc_property_t` [Type]

This data type denotes a boolean property on Unicode characters. It is an immediate type that can be copied by simple assignment, without involving memory allocation. It is not an array type.

Many Unicode properties are predefined.

The following are general properties.

`uc_property_t UC_PROPERTY_WHITE_SPACE` [Constant]
`uc_property_t UC_PROPERTY_ALPHABETIC` [Constant]
`uc_property_t UC_PROPERTY_OTHER_ALPHABETIC` [Constant]
`uc_property_t UC_PROPERTY_NOT_A_CHARACTER` [Constant]
`uc_property_t UC_PROPERTY_DEFAULT_IGNORABLE_CODE_POINT` [Constant]
`uc_property_t UC_PROPERTY_OTHER_DEFAULT_IGNORABLE_CODE_POINT` [Constant]
`uc_property_t UC_PROPERTY_DEPRECATED` [Constant]
`uc_property_t UC_PROPERTY_LOGICAL_ORDER_EXCEPTION` [Constant]
`uc_property_t UC_PROPERTY_VARIATION_SELECTOR` [Constant]
`uc_property_t UC_PROPERTY_PRIVATE_USE` [Constant]
`uc_property_t UC_PROPERTY_UNASSIGNED_CODE_VALUE` [Constant]

The following properties are related to case folding.

`uc_property_t UC_PROPERTY_UPPERCASE` [Constant]
`uc_property_t UC_PROPERTY_OTHER_UPPERCASE` [Constant]
`uc_property_t UC_PROPERTY_LOWERCASE` [Constant]
`uc_property_t UC_PROPERTY_OTHER_LOWERCASE` [Constant]
`uc_property_t UC_PROPERTY_TITLECASE` [Constant]
`uc_property_t UC_PROPERTY_CASED` [Constant]
`uc_property_t UC_PROPERTY_CASE_IGNORABLE` [Constant]
`uc_property_t UC_PROPERTY_CHANGES_WHEN_LOWERCASED` [Constant]
`uc_property_t UC_PROPERTY_CHANGES_WHEN_UPPERCASED` [Constant]
`uc_property_t UC_PROPERTY_CHANGES_WHEN_TITLECASED` [Constant]
`uc_property_t UC_PROPERTY_CHANGES_WHEN_CASEFOLDED` [Constant]
`uc_property_t UC_PROPERTY_CHANGES_WHEN_CASEMAPPED` [Constant]
`uc_property_t UC_PROPERTY_SOFT_DOTTED` [Constant]

The following properties are related to identifiers.

uc_property_t UC_PROPERTY_ID_START [Constant]
uc_property_t UC_PROPERTY_OTHER_ID_START [Constant]
uc_property_t UC_PROPERTY_ID_CONTINUE [Constant]
uc_property_t UC_PROPERTY_OTHER_ID_CONTINUE [Constant]
uc_property_t UC_PROPERTY_XID_START [Constant]
uc_property_t UC_PROPERTY_XID_CONTINUE [Constant]
uc_property_t UC_PROPERTY_PATTERN_WHITE_SPACE [Constant]
uc_property_t UC_PROPERTY_PATTERN_SYNTAX [Constant]

The following properties have an influence on shaping and rendering.

uc_property_t UC_PROPERTY_JOIN_CONTROL [Constant]
uc_property_t UC_PROPERTY_GRAPHEME_BASE [Constant]
uc_property_t UC_PROPERTY_GRAPHEME_EXTEND [Constant]
uc_property_t UC_PROPERTY_OTHER_GRAPHEME_EXTEND [Constant]
uc_property_t UC_PROPERTY_GRAPHEME_LINK [Constant]

The following properties relate to bidirectional reordering.

uc_property_t UC_PROPERTY_BIDI_CONTROL [Constant]
uc_property_t UC_PROPERTY_BIDI_LEFT_TO_RIGHT [Constant]
uc_property_t UC_PROPERTY_BIDI_HEBREW_RIGHT_TO_LEFT [Constant]
uc_property_t UC_PROPERTY_BIDI_ARABIC_RIGHT_TO_LEFT [Constant]
uc_property_t UC_PROPERTY_BIDI_EUROPEAN_DIGIT [Constant]
uc_property_t UC_PROPERTY_BIDI_EUR_NUM_SEPARATOR [Constant]
uc_property_t UC_PROPERTY_BIDI_EUR_NUM_TERMINATOR [Constant]
uc_property_t UC_PROPERTY_BIDI_ARABIC_DIGIT [Constant]
uc_property_t UC_PROPERTY_BIDI_COMMON_SEPARATOR [Constant]
uc_property_t UC_PROPERTY_BIDI_BLOCK_SEPARATOR [Constant]
uc_property_t UC_PROPERTY_BIDI_SEGMENT_SEPARATOR [Constant]
uc_property_t UC_PROPERTY_BIDI_WHITESPACE [Constant]
uc_property_t UC_PROPERTY_BIDI_NON_SPACING_MARK [Constant]
uc_property_t UC_PROPERTY_BIDI_BOUNDARY_NEUTRAL [Constant]
uc_property_t UC_PROPERTY_BIDI_PDF [Constant]
uc_property_t UC_PROPERTY_BIDI_EMBEDDING_OR_OVERRIDE [Constant]
uc_property_t UC_PROPERTY_BIDI_OTHER_NEUTRAL [Constant]

The following properties deal with number representations.

uc_property_t UC_PROPERTY_HEX_DIGIT [Constant]
uc_property_t UC_PROPERTY_ASCII_HEX_DIGIT [Constant]

The following properties deal with CJK.

uc_property_t UC_PROPERTY_IDEOGRAPHIC [Constant]
uc_property_t UC_PROPERTY_UNIFIED_IDEOGRAPH [Constant]
uc_property_t UC_PROPERTY_RADICAL [Constant]
uc_property_t UC_PROPERTY_IDS_BINARY_OPERATOR [Constant]
uc_property_t UC_PROPERTY_IDS_TRINARY_OPERATOR [Constant]

The following properties deal with pictographic symbols.

uc_property_t UC_PROPERTY_EMOJI [Constant]
uc_property_t UC_PROPERTY_EMOJI_PRESENTATION [Constant]

uc_property_t UC_PROPERTY_EMOJI_MODIFIER [Constant]
uc_property_t UC_PROPERTY_EMOJI_MODIFIER_BASE [Constant]
uc_property_t UC_PROPERTY_EMOJI_COMPONENT [Constant]
uc_property_t UC_PROPERTY_EXTENDED_PICTOGRAPHIC [Constant]

Other miscellaneous properties are:

uc_property_t UC_PROPERTY_ZERO_WIDTH [Constant]
uc_property_t UC_PROPERTY_SPACE [Constant]
uc_property_t UC_PROPERTY_NON_BREAK [Constant]
uc_property_t UC_PROPERTY_ISO_CONTROL [Constant]
uc_property_t UC_PROPERTY_FORMAT_CONTROL [Constant]
uc_property_t UC_PROPERTY_DASH [Constant]
uc_property_t UC_PROPERTY_HYPHEN [Constant]
uc_property_t UC_PROPERTY_PUNCTUATION [Constant]
uc_property_t UC_PROPERTY_LINE_SEPARATOR [Constant]
uc_property_t UC_PROPERTY_PARAGRAPH_SEPARATOR [Constant]
uc_property_t UC_PROPERTY_QUOTATION_MARK [Constant]
uc_property_t UC_PROPERTY_SENTENCE_TERMINAL [Constant]
uc_property_t UC_PROPERTY_TERMINAL_PUNCTUATION [Constant]
uc_property_t UC_PROPERTY_CURRENCY_SYMBOL [Constant]
uc_property_t UC_PROPERTY_MATH [Constant]
uc_property_t UC_PROPERTY_OTHER_MATH [Constant]
uc_property_t UC_PROPERTY_PAIRED_PUNCTUATION [Constant]
uc_property_t UC_PROPERTY_LEFT_OF_PAIR [Constant]
uc_property_t UC_PROPERTY_COMBINING [Constant]
uc_property_t UC_PROPERTY_COMPOSITE [Constant]
uc_property_t UC_PROPERTY_DECIMAL_DIGIT [Constant]
uc_property_t UC_PROPERTY_NUMERIC [Constant]
uc_property_t UC_PROPERTY_DIACRITIC [Constant]
uc_property_t UC_PROPERTY_EXTENDER [Constant]
uc_property_t UC_PROPERTY_IGNORABLE_CONTROL [Constant]
uc_property_t UC_PROPERTY_REGIONAL_INDICATOR [Constant]

The following function looks up a property by its name.

uc_property_t uc_property_byname (*const char *property_name*) [Function]

Returns the property given by name, e.g. "White space". If a property with the given name exists, the result will satisfy the uc_property_is_valid predicate. Otherwise the result will not satisfy this predicate and must not be passed to functions that expect an uc_property_t argument.

This lookup ignores spaces, underscores, or hyphens as word separators, is case-insignificant, and supports the aliases listed in Unicode's PropertyAliases.txt file.

This function references a big table of all predefined properties. Its use can significantly increase the size of your application.

bool uc_property_is_valid (*uc_property_t property*) [Function]

Returns true when the given property is valid, or false otherwise.

The following function views a property as a set of Unicode characters.

bool uc_is_property (*ucs4_t* uc, *uc_property_t* property) [Function]

Tests whether the Unicode character *uc* has the given property.

8.9.2 Properties as functions – the functional API

The following are general properties.

bool uc_is_property_white_space (*ucs4_t* uc) [Function]
bool uc_is_property_alphabetic (*ucs4_t* uc) [Function]
bool uc_is_property_other_alphabetic (*ucs4_t* uc) [Function]
bool uc_is_property_not_a_character (*ucs4_t* uc) [Function]
bool uc_is_property_default_ignorable_code_point (*ucs4_t* uc) [Function]
bool uc_is_property_other_default_ignorable_code_point (*ucs4_t* uc) [Function]
bool uc_is_property_deprecated (*ucs4_t* uc) [Function]
bool uc_is_property_logical_order_exception (*ucs4_t* uc) [Function]
bool uc_is_property_variation_selector (*ucs4_t* uc) [Function]
bool uc_is_property_private_use (*ucs4_t* uc) [Function]
bool uc_is_property_unassigned_code_value (*ucs4_t* uc) [Function]

The following properties are related to case folding.

bool uc_is_property_uppercase (*ucs4_t* uc) [Function]
bool uc_is_property_other_uppercase (*ucs4_t* uc) [Function]
bool uc_is_property_lowercase (*ucs4_t* uc) [Function]
bool uc_is_property_other_lowercase (*ucs4_t* uc) [Function]
bool uc_is_property_titlecase (*ucs4_t* uc) [Function]
bool uc_is_property_cased (*ucs4_t* uc) [Function]
bool uc_is_property_case_ignorable (*ucs4_t* uc) [Function]
bool uc_is_property_changes_when_lowercased (*ucs4_t* uc) [Function]
bool uc_is_property_changes_when_uppercased (*ucs4_t* uc) [Function]
bool uc_is_property_changes_when_titlecased (*ucs4_t* uc) [Function]
bool uc_is_property_changes_when_casefolded (*ucs4_t* uc) [Function]
bool uc_is_property_changes_when_casemapped (*ucs4_t* uc) [Function]
bool uc_is_property_soft_dotted (*ucs4_t* uc) [Function]

The following properties are related to identifiers.

bool uc_is_property_id_start (*ucs4_t* uc) [Function]
bool uc_is_property_other_id_start (*ucs4_t* uc) [Function]
bool uc_is_property_id_continue (*ucs4_t* uc) [Function]
bool uc_is_property_other_id_continue (*ucs4_t* uc) [Function]
bool uc_is_property_xid_start (*ucs4_t* uc) [Function]
bool uc_is_property_xid_continue (*ucs4_t* uc) [Function]
bool uc_is_property_pattern_white_space (*ucs4_t* uc) [Function]
bool uc_is_property_pattern_syntax (*ucs4_t* uc) [Function]

The following properties have an influence on shaping and rendering.

bool uc_is_property_join_control (*ucs4_t* uc) [Function]
bool uc_is_property_grapheme_base (*ucs4_t* uc) [Function]
bool uc_is_property_grapheme_extend (*ucs4_t* uc) [Function]

bool uc_is_property_other_grapheme_extend (*ucs4_t uc*) [Function]
bool uc_is_property_grapheme_link (*ucs4_t uc*) [Function]

The following properties relate to bidirectional reordering.

bool uc_is_property_bidi_control (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_left_to_right (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_hebrew_right_to_left (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_arabic_right_to_left (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_european_digit (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_eur_num_separator (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_eur_num_terminator (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_arabic_digit (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_common_separator (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_block_separator (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_segment_separator (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_whitespace (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_non_spacing_mark (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_boundary_neutral (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_pdf (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_embedding_or_override (*ucs4_t uc*) [Function]
bool uc_is_property_bidi_other_neutral (*ucs4_t uc*) [Function]

The following properties deal with number representations.

bool uc_is_property_hex_digit (*ucs4_t uc*) [Function]
bool uc_is_property_ascii_hex_digit (*ucs4_t uc*) [Function]

The following properties deal with CJK.

bool uc_is_property_ideographic (*ucs4_t uc*) [Function]
bool uc_is_property_unified_ideograph (*ucs4_t uc*) [Function]
bool uc_is_property_radical (*ucs4_t uc*) [Function]
bool uc_is_property_ids_binary_operator (*ucs4_t uc*) [Function]
bool uc_is_property_ids_trinary_operator (*ucs4_t uc*) [Function]

The following properties deal with pictographic symbols.

bool uc_is_property_emoji (*ucs4_t uc*) [Function]
bool uc_is_property_emoji_presentation (*ucs4_t uc*) [Function]
bool uc_is_property_emoji_modifier (*ucs4_t uc*) [Function]
bool uc_is_property_emoji_modifier_base (*ucs4_t uc*) [Function]
bool uc_is_property_emoji_component (*ucs4_t uc*) [Function]
bool uc_is_property_extended_pictographic (*ucs4_t uc*) [Function]

Other miscellaneous properties are:

bool uc_is_property_zero_width (*ucs4_t uc*) [Function]
bool uc_is_property_space (*ucs4_t uc*) [Function]
bool uc_is_property_non_break (*ucs4_t uc*) [Function]
bool uc_is_property_iso_control (*ucs4_t uc*) [Function]
bool uc_is_property_format_control (*ucs4_t uc*) [Function]
bool uc_is_property_dash (*ucs4_t uc*) [Function]
bool uc_is_property_hyphen (*ucs4_t uc*) [Function]

`bool uc_is_property_punctuation` (*ucs4_t uc*) [Function]
`bool uc_is_property_line_separator` (*ucs4_t uc*) [Function]
`bool uc_is_property_paragraph_separator` (*ucs4_t uc*) [Function]
`bool uc_is_property_quotation_mark` (*ucs4_t uc*) [Function]
`bool uc_is_property_sentence_terminal` (*ucs4_t uc*) [Function]
`bool uc_is_property_terminal_punctuation` (*ucs4_t uc*) [Function]
`bool uc_is_property_currency_symbol` (*ucs4_t uc*) [Function]
`bool uc_is_property_math` (*ucs4_t uc*) [Function]
`bool uc_is_property_other_math` (*ucs4_t uc*) [Function]
`bool uc_is_property_paired_punctuation` (*ucs4_t uc*) [Function]
`bool uc_is_property_left_of_pair` (*ucs4_t uc*) [Function]
`bool uc_is_property_combining` (*ucs4_t uc*) [Function]
`bool uc_is_property_composite` (*ucs4_t uc*) [Function]
`bool uc_is_property_decimal_digit` (*ucs4_t uc*) [Function]
`bool uc_is_property_numeric` (*ucs4_t uc*) [Function]
`bool uc_is_property_diacritic` (*ucs4_t uc*) [Function]
`bool uc_is_property_extender` (*ucs4_t uc*) [Function]
`bool uc_is_property_ignorable_control` (*ucs4_t uc*) [Function]
`bool uc_is_property_regional_indicator` (*ucs4_t uc*) [Function]

8.10 Scripts

The Unicode characters are subdivided into scripts.

The following type is used to represent a script:

`uc_script_t` [Type]

This data type is a structure type that refers to statically allocated read-only data. It contains the following fields:

```
const char *name;
```

The `name` field contains the name of the script.

The following functions look up a script.

`const uc_script_t * uc_script` (*ucs4_t uc*) [Function]

Returns the script of a Unicode character. Returns NULL if *uc* does not belong to any script.

`const uc_script_t * uc_script_byname` (*const char *script_name*) [Function]

Returns the script given by its name, e.g. `"HAN"`. Returns NULL if a script with the given name does not exist.

The following function views a script as a set of Unicode characters.

`bool uc_is_script` (*ucs4_t uc, const uc_script_t *script*) [Function]

Tests whether a Unicode character belongs to a given script.

The following gives a global picture of all scripts.

void uc_all_scripts (*const uc_script_t* ***scripts*, *size_t* **count*) [Function]

Get the list of all scripts. Stores a pointer to an array of all scripts in **scripts* and the length of this array in **count*.

8.11 Blocks

The Unicode characters are subdivided into blocks. A block is an interval of Unicode code points.

The following type is used to represent a block.

uc_block_t [Type]

This data type is a structure type that refers to statically allocated data. It contains the following fields:

```
ucs4_t start;
ucs4_t end;
const char *name;
```

The `start` field is the first Unicode code point in the block.

The `end` field is the last Unicode code point in the block.

The `name` field is the name of the block.

The following function looks up a block.

const uc_block_t * uc_block (*ucs4_t* *uc*) [Function]

Returns the block a character belongs to.

The following function views a block as a set of Unicode characters.

bool uc_is_block (*ucs4_t* *uc*, *const uc_block_t* **block*) [Function]

Tests whether a Unicode character belongs to a given block.

The following gives a global picture of all block.

void uc_all_blocks (*const uc_block_t* ***blocks*, *size_t* **count*) [Function]

Get the list of all blocks. Stores a pointer to an array of all blocks in **blocks* and the length of this array in **count*.

8.12 ISO C and Java syntax

The following properties are taken from language standards. The supported language standards are ISO C 99 and Java.

bool uc_is_c_whitespace (*ucs4_t* *uc*) [Function]

Tests whether a Unicode character is considered whitespace in ISO C 99.

bool uc_is_java_whitespace (*ucs4_t* *uc*) [Function]

Tests whether a Unicode character is considered whitespace in Java.

The following enumerated values are the possible return values of the functions `uc_c_ident_category` and `uc_java_ident_category`.

`int UC_IDENTIFIER_START` [Constant]
: This return value means that the given character is valid as first or subsequent character in an identifier.

`int UC_IDENTIFIER_VALID` [Constant]
: This return value means that the given character is valid as subsequent character only.

`int UC_IDENTIFIER_INVALID` [Constant]
: This return value means that the given character is not valid in an identifier.

`int UC_IDENTIFIER_IGNORABLE` [Constant]
: This return value (only for Java) means that the given character is ignorable.

The following function determine whether a given character can be a constituent of an identifier in the given programming language.

`int uc_c_ident_category` (*ucs4_t* `uc`) [Function]
: Returns the categorization of a Unicode character with respect to the ISO C 99 identifier syntax.

`int uc_java_ident_category` (*ucs4_t* `uc`) [Function]
: Returns the categorization of a Unicode character with respect to the Java identifier syntax.

8.13 Classifications like in ISO C

The following character classifications mimic those declared in the ISO C header files `<ctype.h>` and `<wctype.h>`. These functions are deprecated, because this set of functions was designed with ASCII in mind and cannot reflect the more diverse reality of the Unicode character set. But they can be a quick-and-dirty porting aid when migrating from `wchar_t` APIs to Unicode strings.

`bool uc_is_alnum` (*ucs4_t* `uc`) [Function]
: Tests for any character for which `uc_is_alpha` or `uc_is_digit` is true.

`bool uc_is_alpha` (*ucs4_t* `uc`) [Function]
: Tests for any character for which `uc_is_upper` or `uc_is_lower` is true, or any character that is one of a locale-specific set of characters for which none of `uc_is_cntrl`, `uc_is_digit`, `uc_is_punct`, or `uc_is_space` is true.

`bool uc_is_cntrl` (*ucs4_t* `uc`) [Function]
: Tests for any control character.

`bool uc_is_digit` (*ucs4_t* `uc`) [Function]
: Tests for any character that corresponds to a decimal-digit character.

`bool uc_is_graph` (*ucs4_t* `uc`) [Function]
: Tests for any character for which `uc_is_print` is true and `uc_is_space` is false.

`bool uc_is_lower (`*ucs4_t uc*`)` [Function]
: Tests for any character that corresponds to a lowercase letter or is one of a locale-specific set of characters for which none of `uc_is_cntrl`, `uc_is_digit`, `uc_is_punct`, or `uc_is_space` is true.

`bool uc_is_print (`*ucs4_t uc*`)` [Function]
: Tests for any printing character.

`bool uc_is_punct (`*ucs4_t uc*`)` [Function]
: Tests for any printing character that is one of a locale-specific set of characters for which neither `uc_is_space` nor `uc_is_alnum` is true.

`bool uc_is_space (`*ucs4_t uc*`)` [Function]
: Test for any character that corresponds to a locale-specific set of characters for which none of `uc_is_alnum`, `uc_is_graph`, or `uc_is_punct` is true.

`bool uc_is_upper (`*ucs4_t uc*`)` [Function]
: Tests for any character that corresponds to an uppercase letter or is one of a locale-specific set of characters for which none of `uc_is_cntrl`, `uc_is_digit`, `uc_is_punct`, or `uc_is_space` is true.

`bool uc_is_xdigit (`*ucs4_t uc*`)` [Function]
: Tests for any character that corresponds to a hexadecimal-digit character.

`bool uc_is_blank (`*ucs4_t uc*`)` [Function]
: Tests for any character that corresponds to a standard blank character or a locale-specific set of characters for which `uc_is_alnum` is false.

9 Display width <uniwidth.h>

This include file declares functions that return the display width, measured in columns, of characters or strings, when output to a device that uses non-proportional fonts.

Note that for some rarely used characters the actual fonts or terminal emulators can use a different width. There is no mechanism for communicating the display width of characters across a Unix pseudo-terminal (tty). Also, there are scripts with complex rendering, like the Indic scripts. For these scripts, there is no such concept as non-proportional fonts. Therefore the results of these functions usually work fine on most scripts and on most characters but can fail to represent the actual display width.

These functions are locale dependent. The *encoding* argument identifies the encoding (e.g. `"ISO-8859-2"` for Polish).

`int uc_width` (*ucs4_t* `uc`, *const char* *`encoding`) [Function]

Determines and returns the number of column positions required for *uc*. Returns -1 if *uc* is a control character that has an influence on the column position when output.

`int u8_width` (*const uint8_t* *`s`, *size_t* `n`, *const char* *`encoding`) [Function]
`int u16_width` (*const uint16_t* *`s`, *size_t* `n`, *const char* *`encoding`) [Function]
`int u32_width` (*const uint32_t* *`s`, *size_t* `n`, *const char* *`encoding`) [Function]

Determines and returns the number of column positions required for first *n* units (or fewer if *s* ends before this) in *s*. This function ignores control characters in the string.

`int u8_strwidth` (*const uint8_t* *`s`, *const char* *`encoding`) [Function]
`int u16_strwidth` (*const uint16_t* *`s`, *const char* *`encoding`) [Function]
`int u32_strwidth` (*const uint32_t* *`s`, *const char* *`encoding`) [Function]

Determines and returns the number of column positions required for *s*. This function ignores control characters in the string.

10 Grapheme cluster breaks in strings `<unigbrk.h>`

This include file declares functions for determining where in a string "grapheme clusters" start and end. A "grapheme cluster" is an approximation to a user-perceived character, which sometimes corresponds to multiple Unicode characters. Editing operations such as mouse selection, cursor movement, and backspacing often operate on grapheme clusters as units, not on individual characters.

Some grapheme clusters are built from a base character and a combining character. The letter 'é', for example, is most commonly represented in Unicode as a single character U+00E8 LATIN SMALL LETTER E WITH ACUTE. It is, however, equally valid to use the pair of characters U+0065 LATIN SMALL LETTER E followed by U+0301 COMBINING ACUTE ACCENT. Since the user would perceive this pair of characters as a single character, they would be grouped into a single grapheme cluster.

But there are also grapheme clusters that consist of several base characters. For example, a Devanagari letter and a Devanagari vowel sign that follows it may form a grapheme cluster. Similarly, some pairs of Thai characters and Hangul syllables (formed by two or three Hangul characters) are grapheme clusters.

10.1 Grapheme cluster breaks in a string

The following functions find a single boundary between grapheme clusters in a string.

`void u8_grapheme_next` (*const uint8_t *`s`, *const uint8_t *`end`) [Function]
`void u16_grapheme_next` (*const uint16_t *`s`, *const uint16_t *`end`) [Function]
`void u32_grapheme_next` (*const uint32_t *`s`, *const uint32_t *`end`) [Function]

Returns the start of the next grapheme cluster following *s*, or *end* if no grapheme cluster break is encountered before it. Returns NULL if and only if `s == end`.

Note that these functions do not handle the case when a character outside of the range between *s* and *end* is needed to determine the boundary. Use `_grapheme_breaks` functions for such cases.

`void u8_grapheme_prev` (*const uint8_t *`s`, *const uint8_t *`start`) [Function]
`void u16_grapheme_prev` (*const uint16_t *`s`, *const uint16_t *`start`) [Function]
`void u32_grapheme_prev` (*const uint32_t *`s`, *const uint32_t *`start`) [Function]

Returns the start of the grapheme cluster preceding *s*, or *start* if no grapheme cluster break is encountered before it. Returns NULL if and only if `s == start`.

Note that these functions do not handle the case when a character outside of the range between *start* and *s* is needed to determine the boundary. Use `_grapheme_breaks` functions for such cases.

Note also that these functions work only on well-formed Unicode strings.

The following functions determine all of the grapheme cluster boundaries in a string.

`void u8_grapheme_breaks` (*const uint8_t *`s`, *size_t* `n`, *char *`p`) [Function]
`void u16_grapheme_breaks` (*const uint16_t *`s`, *size_t* `n`, *char *`p`) [Function]
`void u32_grapheme_breaks` (*const uint32_t *`s`, *size_t* `n`, *char *`p`) [Function]
`void ulc_grapheme_breaks` (*const char *`s`, *size_t* `n`, *char *`p`) [Function]

void uc_grapheme_breaks (*const ucs_t* *s, *size_t* n, *char* *p) [Function]

Determines the grapheme cluster break points in *s*, an array of *n* units, and stores the result at *p*[0..*n*x-1].

p[i] = 1 means that there is a grapheme cluster boundary between *s*[i-1] and *s*[i].

p[i] = 0 means that *s*[i-1] and *s*[i] are part of the same grapheme cluster.

p[0] is always set to 1, because there is always a grapheme cluster break at start of text.

In addition to the above variants for UTF-8, UTF-16, and UTF-32 strings, <unigbrk.h> provides another variant: uc_grapheme_breaks.

This is similar to u32_grapheme_breaks, but it accepts any characters which may not be represented in UTF-32, such as control characters.

10.2 Grapheme cluster break property

This is a more low-level API. The grapheme cluster break property is a property defined in Unicode Standard Annex #29, section "Grapheme Cluster Boundaries", see https://www.unicode.org/reports/tr29/#Grapheme_Cluster_Boundaries. It is used for determining the grapheme cluster breaks in a string.

The following are the possible values of the grapheme cluster break property. More values may be added in the future.

int GBP_OTHER [Constant]
int GBP_CR [Constant]
int GBP_LF [Constant]
int GBP_CONTROL [Constant]
int GBP_EXTEND [Constant]
int GBP_PREPEND [Constant]
int GBP_SPACINGMARK [Constant]
int GBP_L [Constant]
int GBP_V [Constant]
int GBP_T [Constant]
int GBP_LV [Constant]
int GBP_LVT [Constant]
int GBP_RI [Constant]
int GBP_ZWJ [Constant]
int GBP_EB [Constant]
int GBP_EM [Constant]
int GBP_GAZ [Constant]
int GBP_EBG [Constant]

The following function looks up the grapheme cluster break property of a character.

int uc_graphemeclusterbreak_property (*ucs4_t* uc) [Function]

Returns the Grapheme_Cluster_Break property of a Unicode character.

The following function determines whether there is a grapheme cluster break between two Unicode characters. It is the primitive upon which the higher-level functions in the previous section are directly based.

`bool uc_is_grapheme_break (`*ucs4_t* `a`, *ucs4_t* `b)` [Function]

Returns true if there is an grapheme cluster boundary between Unicode characters *a* and *b*.

There is always a grapheme cluster break at the start or end of text. You can specify zero for *a* or *b* to indicate start of text or end of text, respectively.

This implements the extended (not legacy) grapheme cluster rules described in the Unicode standard, because the standard says that they are preferred.

Note that this function does not handle the case when three or more consecutive characters are needed to determine the boundary. Use `uc_grapheme_breaks` for such cases.

11 Word breaks in strings <uniwbrk.h>

This include file declares functions for determining where in a string "words" start and end. Here "words" are not necessarily the same as entities that can be looked up in dictionaries, but rather groups of consecutive characters that should not be split by text processing operations.

11.1 Word breaks in a string

The following functions determine the word breaks in a string.

void u8_wordbreaks (*const uint8_t *s, size_t n, char *p*) [Function]
void u16_wordbreaks (*const uint16_t *s, size_t n, char *p*) [Function]
void u32_wordbreaks (*const uint32_t *s, size_t n, char *p*) [Function]
void ulc_wordbreaks (*const char *s, size_t n, char *p*) [Function]

Determines the word break points in *s*, an array of *n* units, and stores the result at `p[0..n-1]`.

`p[i] = 1` means that there is a word boundary between `s[i-1]` and `s[i]`.

`p[i] = 0` means that `s[i-1]` and `s[i]` must not be separated.

`p[0]` is always set to 0. If an application wants to consider a word break to be present at the beginning of the string (before `s[0]`) or at the end of the string (after `s[0..n-1]`), it has to treat these cases explicitly.

11.2 Word break property

This is a more low-level API. The word break property is a property defined in Unicode Standard Annex #29, section "Word Boundaries", see `https://www.unicode.org/reports/tr29/#Word_Boundaries`. It is used for determining the word breaks in a string.

The following are the possible values of the word break property. More values may be added in the future.

int WBP_OTHER [Constant]
int WBP_CR [Constant]
int WBP_LF [Constant]
int WBP_NEWLINE [Constant]
int WBP_EXTEND [Constant]
int WBP_FORMAT [Constant]
int WBP_KATAKANA [Constant]
int WBP_ALETTER [Constant]
int WBP_MIDNUMLET [Constant]
int WBP_MIDLETTER [Constant]
int WBP_MIDNUM [Constant]
int WBP_NUMERIC [Constant]
int WBP_EXTENDNUMLET [Constant]
int WBP_RI [Constant]
int WBP_DQ [Constant]
int WBP_SQ [Constant]

`int WBP_HL` [Constant]
`int WBP_ZWJ` [Constant]
`int WBP_EB` [Constant]
`int WBP_EM` [Constant]
`int WBP_GAZ` [Constant]
`int WBP_EBG` [Constant]

The following function looks up the word break property of a character.

`int uc_wordbreak_property` (*ucs4_t uc*) [Function]

Returns the Word_Break property of a Unicode character.

12 Line breaking <unilbrk.h>

This include file declares functions for determining where in a string line breaks could or should be introduced, in order to make the displayed string fit into a column of given width.

These functions are locale dependent. The *encoding* argument identifies the encoding (e.g. `"ISO-8859-2"` for Polish).

The following enumerated values indicate whether, at a given position, a line break is possible or not. Given an string *s* as an array `s[0..n-1]` and a position *i*, the values have the following meanings:

`int UC_BREAK_MANDATORY` [Constant]
: This value indicates that `s[i]` is a line break character.

`int UC_BREAK_CR_BEFORE_LF` [Constant]
: This value is a variant of `UC_BREAK_MANDATORY`. It indicates that `s[i]` is a CR character and that `s[i+1]` is a LF character.

`int UC_BREAK_POSSIBLE` [Constant]
: This value indicates that a line break may be inserted between `s[i-1]` and `s[i]`.

`int UC_BREAK_HYPHENATION` [Constant]
: This value indicates that a hyphen and a line break may be inserted between `s[i-1]` and `s[i]`. But beware of language dependent hyphenation rules.

`int UC_BREAK_PROHIBITED` [Constant]
: This value indicates that `s[i-1]` and `s[i]` must not be separated.

`int UC_BREAK_UNDEFINED` [Constant]
: This value is not used as a return value; rather, in the overriding argument of the `u*_width_linebreaks` functions, it indicates the absence of an override.

The following functions determine the positions at which line breaks are possible.

`void u8_possible_linebreaks` (*const uint8_t *s*, *size_t n*, *const char *encoding*, *char *p*) [Function]
`void u16_possible_linebreaks` (*const uint16_t *s*, *size_t n*, *const char *encoding*, *char *p*) [Function]
`void u32_possible_linebreaks` (*const uint32_t *s*, *size_t n*, *const char *encoding*, *char *p*) [Function]
`void ulc_possible_linebreaks` (*const char *s*, *size_t n*, *const char *encoding*, *char *p*) [Function]
: Determines the line break points in *s*, and stores the result at `p[0..n-1]`. Every `p[i]` is assigned one of the values `UC_BREAK_MANDATORY`, `UC_BREAK_CR_BEFORE_LF`, `UC_BREAK_POSSIBLE`, `UC_BREAK_HYPHENATION`, `UC_BREAK_PROHIBITED`.

The following functions determine where line breaks should be inserted so that each line fits in a given width, when output to a device that uses non-proportional fonts.

`int u8_width_linebreaks` (*const uint8_t* *`s`, *size_t* `n`, *int* `width`, *int* `start_column`, *int* `at_end_columns`, *const char* *`override`, *const char* *`encoding`, *char* *`p`) [Function]

`int u16_width_linebreaks` (*const uint16_t* *`s`, *size_t* `n`, *int* `width`, *int* `start_column`, *int* `at_end_columns`, *const char* *`override`, *const char* *`encoding`, *char* *`p`) [Function]

`int u32_width_linebreaks` (*const uint32_t* *`s`, *size_t* `n`, *int* `width`, *int* `start_column`, *int* `at_end_columns`, *const char* *`override`, *const char* *`encoding`, *char* *`p`) [Function]

`int ulc_width_linebreaks` (*const char* *`s`, *size_t* `n`, *int* `width`, *int* `start_column`, *int* `at_end_columns`, *const char* *`override`, *const char* *`encoding`, *char* *`p`) [Function]

Chooses the best line breaks, assuming that every character occupies a width given by the `uc_width` function (see Chapter 9 [uniwidth.h], page 49).

The string is `s[0..n-1]`.

The maximum number of columns per line is given as *width*. The starting column of the string is given as *start_column*. If the algorithm shall keep room after the last piece, this amount of room can be given as *at_end_columns*.

override is an optional override; if `override[i] != UC_BREAK_UNDEFINED`, `override[i]` takes precedence over `p[i]` as returned by the `u*_possible_linebreaks` function.

The given *encoding* is used for disambiguating widths in `uc_width`.

Returns the column after the end of the string, and stores the result at `p[0..n-1]`. Every `p[i]` is assigned one of the values `UC_BREAK_MANDATORY`, `UC_BREAK_CR_BEFORE_LF`, `UC_BREAK_POSSIBLE`, `UC_BREAK_HYPHENATION`, `UC_BREAK_PROHIBITED`. Here the value `UC_BREAK_POSSIBLE` indicates that a line break *should* be inserted.

13 Normalization forms (composition and decomposition) <uninorm.h>

This include file defines functions for transforming Unicode strings to one of the four normal forms, known as NFC, NFD, NKFC, NFKD. These transformations involve decomposition and — for NFC and NFKC — composition of Unicode characters.

13.1 Decomposition of Unicode characters

The following enumerated values are the possible types of decomposition of a Unicode character.

`int UC_DECOMP_CANONICAL` [Constant]
Denotes canonical decomposition.

`int UC_DECOMP_FONT` [Constant]
UCD marker: `<font>`. Denotes a font variant (e.g. a blackletter form).

`int UC_DECOMP_NOBREAK` [Constant]
UCD marker: `<noBreak>`. Denotes a no-break version of a space or hyphen.

`int UC_DECOMP_INITIAL` [Constant]
UCD marker: `<initial>`. Denotes an initial presentation form (Arabic).

`int UC_DECOMP_MEDIAL` [Constant]
UCD marker: `<medial>`. Denotes a medial presentation form (Arabic).

`int UC_DECOMP_FINAL` [Constant]
UCD marker: `<final>`. Denotes a final presentation form (Arabic).

`int UC_DECOMP_ISOLATED` [Constant]
UCD marker: `<isolated>`. Denotes an isolated presentation form (Arabic).

`int UC_DECOMP_CIRCLE` [Constant]
UCD marker: `<circle>`. Denotes an encircled form.

`int UC_DECOMP_SUPER` [Constant]
UCD marker: `<super>`. Denotes a superscript form.

`int UC_DECOMP_SUB` [Constant]
UCD marker: `<sub>`. Denotes a subscript form.

`int UC_DECOMP_VERTICAL` [Constant]
UCD marker: `<vertical>`. Denotes a vertical layout presentation form.

`int UC_DECOMP_WIDE` [Constant]
UCD marker: `<wide>`. Denotes a wide (or zenkaku) compatibility character.

`int UC_DECOMP_NARROW` [Constant]
UCD marker: `<narrow>`. Denotes a narrow (or hankaku) compatibility character.

`int UC_DECOMP_SMALL` [Constant]
UCD marker: `<small>`. Denotes a small variant form (CNS compatibility).

`int UC_DECOMP_SQUARE` [Constant]
: UCD marker: `<square>`. Denotes a CJK squared font variant.

`int UC_DECOMP_FRACTION` [Constant]
: UCD marker: `<fraction>`. Denotes a vulgar fraction form.

`int UC_DECOMP_COMPAT` [Constant]
: UCD marker: `<compat>`. Denotes an otherwise unspecified compatibility character.

The following constant denotes the maximum size of decomposition of a single Unicode character.

`unsigned int UC_DECOMPOSITION_MAX_LENGTH` [Macro]
: This macro expands to a constant that is the required size of buffer passed to the `uc_decomposition` and `uc_canonical_decomposition` functions.

The following functions decompose a Unicode character.

`int uc_decomposition` (*ucs4_t* `uc`, *int* `*decomp_tag`, *ucs4_t* `*decomposition`) [Function]
: Returns the character decomposition mapping of the Unicode character *uc*. *decomposition* must point to an array of at least `UC_DECOMPOSITION_MAX_LENGTH` `ucs_t` elements.

  When a decomposition exists, `decomposition[0..n-1]` and `*decomp_tag` are filled and *n* is returned. Otherwise -1 is returned.

`int uc_canonical_decomposition` (*ucs4_t* `uc`, *ucs4_t* `*decomposition`) [Function]
: Returns the canonical character decomposition mapping of the Unicode character *uc*. *decomposition* must point to an array of at least `UC_DECOMPOSITION_MAX_LENGTH` `ucs_t` elements.

  When a decomposition exists, `decomposition[0..n-1]` is filled and *n* is returned. Otherwise -1 is returned.

  Note: This function returns the (simple) "canonical decomposition" of *uc*. If you want the "full canonical decomposition" of *uc*, that is, the recursive application of "canonical decomposition", use the function `u*_normalize` with argument `UNINORM_NFD` instead.

13.2 Composition of Unicode characters

The following function composes a Unicode character from two Unicode characters.

`ucs4_t uc_composition` (*ucs4_t* `uc1`, *ucs4_t* `uc2`) [Function]
: Attempts to combine the Unicode characters *uc1*, *uc2*. *uc1* is known to have canonical combining class 0.

  Returns the combination of *uc1* and *uc2*, if it exists. Returns 0 otherwise.

  Not all decompositions can be recombined using this function. See the Unicode file `CompositionExclusions.txt` for details.

13.3 Normalization of strings

The Unicode standard defines four normalization forms for Unicode strings. The following type is used to denote a normalization form.

`uninorm_t` [Type]

An object of type `uninorm_t` denotes a Unicode normalization form. This is a scalar type; its values can be compared with `==`.

The following constants denote the four normalization forms.

`uninorm_t UNINORM_NFD` [Macro]

Denotes Normalization form D: canonical decomposition.

`uninorm_t UNINORM_NFC` [Macro]

Normalization form C: canonical decomposition, then canonical composition.

`uninorm_t UNINORM_NFKD` [Macro]

Normalization form KD: compatibility decomposition.

`uninorm_t UNINORM_NFKC` [Macro]

Normalization form KC: compatibility decomposition, then canonical composition.

The following functions operate on `uninorm_t` objects.

`bool uninorm_is_compat_decomposing` (*uninorm_t* `nf`) [Function]

Tests whether the normalization form *nf* does compatibility decomposition.

`bool uninorm_is_composing` (*uninorm_t* `nf`) [Function]

Tests whether the normalization form *nf* includes canonical composition.

`uninorm_t uninorm_decomposing_form` (*uninorm_t* `nf`) [Function]

Returns the decomposing variant of the normalization form *nf*. This maps NFC,NFD → NFD and NFKC,NFKD → NFKD.

The following functions apply a Unicode normalization form to a Unicode string.

`uint8_t * u8_normalize` (*uninorm_t* `nf`, *const uint8_t* *`s`, *size_t* `n`, *uint8_t* *`resultbuf`, *size_t* *`lengthp`) [Function]

`uint16_t * u16_normalize` (*uninorm_t* `nf`, *const uint16_t* *`s`, *size_t* `n`, *uint16_t* *`resultbuf`, *size_t* *`lengthp`) [Function]

`uint32_t * u32_normalize` (*uninorm_t* `nf`, *const uint32_t* *`s`, *size_t* `n`, *uint32_t* *`resultbuf`, *size_t* *`lengthp`) [Function]

Returns the specified normalization form of a string.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

13.4 Normalizing comparisons

The following functions compare Unicode string, ignoring differences in normalization.

`int u8_normcmp` (*const uint8_t *s1, size_t n1, const uint8_t *s2, size_t n2, uninorm_t nf, int *resultp*) [Function]
`int u16_normcmp` (*const uint16_t *s1, size_t n1, const uint16_t *s2, size_t n2, uninorm_t nf, int *resultp*) [Function]
`int u32_normcmp` (*const uint32_t *s1, size_t n1, const uint32_t *s2, size_t n2, uninorm_t nf, int *resultp*) [Function]

Compares *s1* and *s2*, ignoring differences in normalization.

nf must be either `UNINORM_NFD` or `UNINORM_NFKD`.

If successful, sets `*resultp` to -1 if *s1* < *s2*, 0 if *s1* = *s2*, 1 if *s1* > *s2*, and returns 0. Upon failure, returns -1 with `errno` set.

`char * u8_normxfrm` (*const uint8_t *s, size_t n, uninorm_t nf, char *resultbuf, size_t *lengthp*) [Function]
`char * u16_normxfrm` (*const uint16_t *s, size_t n, uninorm_t nf, char *resultbuf, size_t *lengthp*) [Function]
`char * u32_normxfrm` (*const uint32_t *s, size_t n, uninorm_t nf, char *resultbuf, size_t *lengthp*) [Function]

Converts the string *s* of length *n* to a NUL-terminated byte sequence, in such a way that comparing `u8_normxfrm (s1)` and `u8_normxfrm (s2)` with the `u8_cmp2` function is equivalent to comparing *s1* and *s2* with the `u8_normcoll` function.

nf must be either `UNINORM_NFC` or `UNINORM_NFKC`.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

`int u8_normcoll` (*const uint8_t *s1, size_t n1, const uint8_t *s2, size_t n2, uninorm_t nf, int *resultp*) [Function]
`int u16_normcoll` (*const uint16_t *s1, size_t n1, const uint16_t *s2, size_t n2, uninorm_t nf, int *resultp*) [Function]
`int u32_normcoll` (*const uint32_t *s1, size_t n1, const uint32_t *s2, size_t n2, uninorm_t nf, int *resultp*) [Function]

Compares *s1* and *s2*, ignoring differences in normalization, using the collation rules of the current locale.

nf must be either `UNINORM_NFC` or `UNINORM_NFKC`.

If successful, sets `*resultp` to -1 if *s1* < *s2*, 0 if *s1* = *s2*, 1 if *s1* > *s2*, and returns 0. Upon failure, returns -1 with `errno` set.

13.5 Normalization of streams of Unicode characters

A "stream of Unicode characters" is essentially a function that accepts an `ucs4_t` argument repeatedly, optionally combined with a function that "flushes" the stream.

`struct uninorm_filter` [Type]

This is the data type of a stream of Unicode characters that normalizes its input according to a given normalization form and passes the normalized character sequence to the encapsulated stream of Unicode characters.

`struct uninorm_filter * uninorm_filter_create` (*uninorm_t* `nf`, *int* (*`stream_func`) (*void* *`stream_data`, *ucs4_t* `uc`), *void* *`stream_data`) [Function]

Creates and returns a normalization filter for Unicode characters.

The pair (*stream_func*, *stream_data*) is the encapsulated stream. `stream_func (stream_data, uc)` receives the Unicode character *uc* and returns 0 if successful, or -1 with `errno` set upon failure.

Returns the new filter, or NULL with `errno` set upon failure.

`int uninorm_filter_write` (*struct uninorm_filter* *`filter`, *ucs4_t* `uc`) [Function]

Stuffs a Unicode character into a normalizing filter. Returns 0 if successful, or -1 with `errno` set upon failure.

`int uninorm_filter_flush` (*struct uninorm_filter* *`filter`) [Function]

Brings data buffered in the filter to its destination, the encapsulated stream.

Returns 0 if successful, or -1 with `errno` set upon failure.

Note! If after calling this function, additional characters are written into the filter, the resulting character sequence in the encapsulated stream will not necessarily be normalized.

`int uninorm_filter_free` (*struct uninorm_filter* *`filter`) [Function]

Brings data buffered in the filter to its destination, the encapsulated stream, then closes and frees the filter.

Returns 0 if successful, or -1 with `errno` set upon failure.

14 Case mappings <unicase.h>

This include file defines functions for case mapping for Unicode strings and case insensitive comparison of Unicode strings and C strings.

These string functions fix the problems that were mentioned in Section 1.5 [char * strings], page 4, namely, they handle the Croatian LETTER DZ WITH CARON, the German LATIN SMALL LETTER SHARP S, the Greek sigma and the Lithuanian i correctly.

14.1 Case mappings of characters

The following functions implement case mappings on Unicode characters — for those cases only where the result of the mapping is a again a single Unicode character.

These mappings are locale and context independent.

> **WARNING!** These functions are not sufficient for languages such as German, Greek and Lithuanian. Better use the functions below that treat an entire string at once and are language aware.

`ucs4_t uc_toupper` (*ucs4_t* `uc`) [Function]

Returns the uppercase mapping of the Unicode character *uc*.

`ucs4_t uc_tolower` (*ucs4_t* `uc`) [Function]

Returns the lowercase mapping of the Unicode character *uc*.

`ucs4_t uc_totitle` (*ucs4_t* `uc`) [Function]

Returns the titlecase mapping of the Unicode character *uc*.

The titlecase mapping of a character is to be used when the character should look like upper case and the following characters are lower cased.

For most characters, this is the same as the uppercase mapping. There are only few characters where the title case variant and the uuper case variant are different. These characters occur in the Latin writing of the Croatian, Bosnian, and Serbian languages.

Lower case	**Title case**	**Upper case**
LATIN SMALL LETTER LJ	LATIN CAPITAL LETTER L WITH SMALL LETTER J	LATIN CAPITAL LETTER LJ
LATIN SMALL LETTER NJ	LATIN CAPITAL LETTER N WITH SMALL LETTER J	LATIN CAPITAL LETTER NJ
LATIN SMALL LETTER DZ	LATIN CAPITAL LETTER D WITH SMALL LETTER Z	LATIN CAPITAL LETTER DZ
LATIN SMALL LETTER DZ WITH CARON	LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON	LATIN CAPITAL LETTER DZ WITH CARON

14.2 Case mappings of strings

Case mapping should always be performed on entire strings, not on individual characters. The functions in this sections do so.

These functions allow to apply a normalization after the case mapping. The reason is that if you want to treat 'ä' and 'Ä' the same, you most often also want to treat the composed and decomposed forms of such a character, U+00C4 LATIN CAPITAL LETTER A WITH DIAERESIS and U+0041 LATIN CAPITAL LETTER A U+0308 COMBINING DIAERESIS the same. The *nf* argument designates the normalization.

These functions are locale dependent. The *iso639_language* argument identifies the language (e.g. `"tr"` for Turkish). NULL means to use locale independent case mappings.

`const char * uc_locale_language ()` [Function]

Returns the ISO 639 language code of the current locale. Returns `""` if it is unknown, or in the `"C"` locale.

`uint8_t * u8_toupper` (*const uint8_t* *`s`, *size_t* `n`, *const char* *`iso639_language`, *uninorm_t* `nf`, *uint8_t* *`resultbuf`, *size_t* *`lengthp`) [Function]
`uint16_t * u16_toupper` (*const uint16_t* *`s`, *size_t* `n`, *const char* *`iso639_language`, *uninorm_t* `nf`, *uint16_t* *`resultbuf`, *size_t* *`lengthp`) [Function]
`uint32_t * u32_toupper` (*const uint32_t* *`s`, *size_t* `n`, *const char* *`iso639_language`, *uninorm_t* `nf`, *uint32_t* *`resultbuf`, *size_t* *`lengthp`) [Function]

Returns the uppercase mapping of a string.

The *nf* argument identifies the normalization form to apply after the case-mapping. It can also be NULL, for no normalization.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

`uint8_t * u8_tolower` (*const uint8_t* *`s`, *size_t* `n`, *const char* *`iso639_language`, *uninorm_t* `nf`, *uint8_t* *`resultbuf`, *size_t* *`lengthp`) [Function]
`uint16_t * u16_tolower` (*const uint16_t* *`s`, *size_t* `n`, *const char* *`iso639_language`, *uninorm_t* `nf`, *uint16_t* *`resultbuf`, *size_t* *`lengthp`) [Function]
`uint32_t * u32_tolower` (*const uint32_t* *`s`, *size_t* `n`, *const char* *`iso639_language`, *uninorm_t* `nf`, *uint32_t* *`resultbuf`, *size_t* *`lengthp`) [Function]

Returns the lowercase mapping of a string.

The *nf* argument identifies the normalization form to apply after the case-mapping. It can also be NULL, for no normalization.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

uint8_t * u8_totitle (*const uint8_t *s*, *size_t n*, *const char *iso639_language*, *uninorm_t nf*, *uint8_t *resultbuf*, *size_t *lengthp*) [Function]

uint16_t * u16_totitle (*const uint16_t *s*, *size_t n*, *const char *iso639_language*, *uninorm_t nf*, *uint16_t *resultbuf*, *size_t *lengthp*) [Function]

uint32_t * u32_totitle (*const uint32_t *s*, *size_t n*, *const char *iso639_language*, *uninorm_t nf*, *uint32_t *resultbuf*, *size_t *lengthp*) [Function]

Returns the titlecase mapping of a string.

Mapping to title case means that, in each word, the first cased character is being mapped to title case and the remaining characters of the word are being mapped to lower case.

The *nf* argument identifies the normalization form to apply after the case-mapping. It can also be NULL, for no normalization.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

14.3 Case mappings of substrings

Case mapping of a substring cannot simply be performed by extracting the substring and then applying the case mapping function to it. This does not work because case mapping requires some information about the surrounding characters. The following functions allow to apply case mappings to substrings of a given string, while taking into account the characters that precede it (the "prefix") and the characters that follow it (the "suffix").

casing_prefix_context_t [Type]

This data type denotes the case-mapping context that is given by a prefix string. It is an immediate type that can be copied by simple assignment, without involving memory allocation. It is not an array type.

casing_prefix_context_t unicase_empty_prefix_context [Constant]

This constant is the case-mapping context that corresponds to an empty prefix string.

The following functions return `casing_prefix_context_t` objects:

casing_prefix_context_t u8_casing_prefix_context (*const uint8_t *s*, *size_t n*) [Function]

casing_prefix_context_t u16_casing_prefix_context (*const uint16_t *s*, *size_t n*) [Function]

casing_prefix_context_t u32_casing_prefix_context (*const uint32_t *s*, *size_t n*) [Function]

Returns the case-mapping context of a given prefix string.

casing_prefix_context_t u8_casing_prefixes_context (*const uint8_t *s*, *size_t n*, *casing_prefix_context_t a_context*) [Function]

casing_prefix_context_t u16_casing_prefixes_context (*const uint16_t *s*, *size_t n*, *casing_prefix_context_t a_context*) [Function]

casing_prefix_context_t u32_casing_prefixes_context [Function]
(*const uint32_t* *s, *size_t* n, *casing_prefix_context_t* a_context)

Returns the case-mapping context of the prefix concat(*a*, *s*), given the case-mapping context of the prefix *a*.

casing_suffix_context_t [Type]

This data type denotes the case-mapping context that is given by a suffix string. It is an immediate type that can be copied by simple assignment, without involving memory allocation. It is not an array type.

casing_suffix_context_t unicase_empty_suffix_context [Constant]

This constant is the case-mapping context that corresponds to an empty suffix string.

The following functions return casing_suffix_context_t objects:

casing_suffix_context_t u8_casing_suffix_context [Function]
(*const uint8_t* *s, *size_t* n)

casing_suffix_context_t u16_casing_suffix_context [Function]
(*const uint16_t* *s, *size_t* n)

casing_suffix_context_t u32_casing_suffix_context [Function]
(*const uint32_t* *s, *size_t* n)

Returns the case-mapping context of a given suffix string.

casing_suffix_context_t u8_casing_suffixes_context [Function]
(*const uint8_t* *s, *size_t* n, *casing_suffix_context_t* a_context)

casing_suffix_context_t u16_casing_suffixes_context [Function]
(*const uint16_t* *s, *size_t* n, *casing_suffix_context_t* a_context)

casing_suffix_context_t u32_casing_suffixes_context [Function]
(*const uint32_t* *s, *size_t* n, *casing_suffix_context_t* a_context)

Returns the case-mapping context of the suffix concat(*s*, a), given the case-mapping context of the suffix *a*.

The following functions perform a case mapping, considering the prefix context and the suffix context.

uint8_t * u8_ct_toupper (*const uint8_t* *s, *size_t* n, [Function]
casing_prefix_context_t prefix_context,
casing_suffix_context_t suffix_context, *const char* *iso639_language,
uninorm_t nf, *uint8_t* *resultbuf, *size_t* *lengthp)

uint16_t * u16_ct_toupper (*const uint16_t* *s, *size_t* n, [Function]
casing_prefix_context_t prefix_context,
casing_suffix_context_t suffix_context, *const char* *iso639_language,
uninorm_t nf, *uint16_t* *resultbuf, *size_t* *lengthp)

uint32_t * u32_ct_toupper (*const uint32_t* *s, *size_t* n, [Function]
casing_prefix_context_t prefix_context,
casing_suffix_context_t suffix_context, *const char* *iso639_language,
uninorm_t nf, *uint32_t* *resultbuf, *size_t* *lengthp)

Returns the uppercase mapping of a string that is surrounded by a prefix and a suffix.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

uint8_t * u8_ct_tolower (*const uint8_t* *s, *size_t* n, *casing_prefix_context_t* prefix_context, *casing_suffix_context_t* suffix_context, *const char* *iso639_language, *uninorm_t* nf, *uint8_t* *resultbuf, *size_t* *lengthp) [Function]

uint16_t * u16_ct_tolower (*const uint16_t* *s, *size_t* n, *casing_prefix_context_t* prefix_context, *casing_suffix_context_t* suffix_context, *const char* *iso639_language, *uninorm_t* nf, *uint16_t* *resultbuf, *size_t* *lengthp) [Function]

uint32_t * u32_ct_tolower (*const uint32_t* *s, *size_t* n, *casing_prefix_context_t* prefix_context, *casing_suffix_context_t* suffix_context, *const char* *iso639_language, *uninorm_t* nf, *uint32_t* *resultbuf, *size_t* *lengthp) [Function]

Returns the lowercase mapping of a string that is surrounded by a prefix and a suffix.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

uint8_t * u8_ct_totitle (*const uint8_t* *s, *size_t* n, *casing_prefix_context_t* prefix_context, *casing_suffix_context_t* suffix_context, *const char* *iso639_language, *uninorm_t* nf, *uint8_t* *resultbuf, *size_t* *lengthp) [Function]

uint16_t * u16_ct_totitle (*const uint16_t* *s, *size_t* n, *casing_prefix_context_t* prefix_context, *casing_suffix_context_t* suffix_context, *const char* *iso639_language, *uninorm_t* nf, *uint16_t* *resultbuf, *size_t* *lengthp) [Function]

uint32_t * u32_ct_totitle (*const uint32_t* *s, *size_t* n, *casing_prefix_context_t* prefix_context, *casing_suffix_context_t* suffix_context, *const char* *iso639_language, *uninorm_t* nf, *uint32_t* *resultbuf, *size_t* *lengthp) [Function]

Returns the titlecase mapping of a string that is surrounded by a prefix and a suffix.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

For example, to uppercase the UTF-8 substring between `s + start_index` and `s + end_index` of a string that extends from `s` to `s + u8_strlen (s)`, you can use the statements

```
size_t result_length;
uint8_t result =
  u8_ct_toupper (s + start_index, end_index - start_index,
                 u8_casing_prefix_context (s, start_index),
                 u8_casing_suffix_context (s + end_index,
                                           u8_strlen (s) - end_index),
                 iso639_language, NULL, NULL, &result_length);
```

14.4 Case insensitive comparison

The following functions implement comparison that ignores differences in case and normalization.

uint8_t * u8_casefold (*const uint8_t *s*, *size_t n*, *const char *iso639_language*, *uninorm_t nf*, *uint8_t *resultbuf*, *size_t *lengthp*) [Function]
uint16_t * u16_casefold (*const uint16_t *s*, *size_t n*, *const char *iso639_language*, *uninorm_t nf*, *uint16_t *resultbuf*, *size_t *lengthp*) [Function]
uint32_t * u32_casefold (*const uint32_t *s*, *size_t n*, *const char *iso639_language*, *uninorm_t nf*, *uint32_t *resultbuf*, *size_t *lengthp*) [Function]

Returns the case folded string.

Comparing `u8_casefold (s1)` and `u8_casefold (s2)` with the `u8_cmp2` function is equivalent to comparing *s1* and *s2* with `u8_casecmp`.

The *nf* argument identifies the normalization form to apply after the case-mapping. It can also be NULL, for no normalization.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

uint8_t * u8_ct_casefold (*const uint8_t *s*, *size_t n*, *casing_prefix_context_t prefix_context*, *casing_suffix_context_t suffix_context*, *const char *iso639_language*, *uninorm_t nf*, *uint8_t *resultbuf*, *size_t *lengthp*) [Function]
uint16_t * u16_ct_casefold (*const uint16_t *s*, *size_t n*, *casing_prefix_context_t prefix_context*, *casing_suffix_context_t suffix_context*, *const char *iso639_language*, *uninorm_t nf*, *uint16_t *resultbuf*, *size_t *lengthp*) [Function]
uint32_t * u32_ct_casefold (*const uint32_t *s*, *size_t n*, *casing_prefix_context_t prefix_context*, *casing_suffix_context_t suffix_context*, *const char *iso639_language*, *uninorm_t nf*, *uint32_t *resultbuf*, *size_t *lengthp*) [Function]

Returns the case folded string. The case folding takes into account the case mapping contexts of the prefix and suffix strings.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

int u8_casecmp (*const uint8_t *s1*, *size_t n1*, *const uint8_t *s2*, *size_t n2*, *const char *iso639_language*, *uninorm_t nf*, *int *resultp*) [Function]
int u16_casecmp (*const uint16_t *s1*, *size_t n1*, *const uint16_t *s2*, *size_t n2*, *const char *iso639_language*, *uninorm_t nf*, *int *resultp*) [Function]
int u32_casecmp (*const uint32_t *s1*, *size_t n1*, *const uint32_t *s2*, *size_t n2*, *const char *iso639_language*, *uninorm_t nf*, *int *resultp*) [Function]
int ulc_casecmp (*const char *s1*, *size_t n1*, *const char *s2*, *size_t n2*, *const char *iso639_language*, *uninorm_t nf*, *int *resultp*) [Function]

Compares *s1* and *s2*, ignoring differences in case and normalization.

The *nf* argument identifies the normalization form to apply after the case-mapping. It can also be NULL, for no normalization.

If successful, sets `*resultp` to -1 if *s1* < *s2*, 0 if *s1* = *s2*, 1 if *s1* > *s2*, and returns 0. Upon failure, returns -1 with `errno` set.

The following functions additionally take into account the sorting rules of the current locale.

char * u8_casexfrm (*const uint8_t* ***s**, *size_t* **n**, *const char* ***iso639_language**, *uninorm_t* **nf**, *char* ***resultbuf**, *size_t* ***lengthp**) [Function]
char * u16_casexfrm (*const uint16_t* ***s**, *size_t* **n**, *const char* ***iso639_language**, *uninorm_t* **nf**, *char* ***resultbuf**, *size_t* ***lengthp**) [Function]
char * u32_casexfrm (*const uint32_t* ***s**, *size_t* **n**, *const char* ***iso639_language**, *uninorm_t* **nf**, *char* ***resultbuf**, *size_t* ***lengthp**) [Function]
char * ulc_casexfrm (*const char* ***s**, *size_t* **n**, *const char* ***iso639_language**, *uninorm_t* **nf**, *char* ***resultbuf**, *size_t* ***lengthp**) [Function]

Converts the string *s* of length *n* to a NUL-terminated byte sequence, in such a way that comparing `u8_casexfrm (s1)` and `u8_casexfrm (s2)` with the gnulib function `memcmp2` is equivalent to comparing *s1* and *s2* with `u8_casecoll`.

nf must be either `UNINORM_NFC`, `UNINORM_NFKC`, or NULL for no normalization.

The *resultbuf* and *lengthp* arguments are as described in chapter Chapter 2 [Conventions], page 7.

int u8_casecoll (*const uint8_t* ***s1**, *size_t* **n1**, *const uint8_t* ***s2**, *size_t* **n2**, *const char* ***iso639_language**, *uninorm_t* **nf**, *int* ***resultp**) [Function]
int u16_casecoll (*const uint16_t* ***s1**, *size_t* **n1**, *const uint16_t* ***s2**, *size_t* **n2**, *const char* ***iso639_language**, *uninorm_t* **nf**, *int* ***resultp**) [Function]
int u32_casecoll (*const uint32_t* ***s1**, *size_t* **n1**, *const uint32_t* ***s2**, *size_t* **n2**, *const char* ***iso639_language**, *uninorm_t* **nf**, *int* ***resultp**) [Function]
int ulc_casecoll (*const char* ***s1**, *size_t* **n1**, *const char* ***s2**, *size_t* **n2**, *const char* ***iso639_language**, *uninorm_t* **nf**, *int* ***resultp**) [Function]

Compares *s1* and *s2*, ignoring differences in case and normalization, using the collation rules of the current locale.

The *nf* argument identifies the normalization form to apply after the case-mapping. It must be either `UNINORM_NFC` or `UNINORM_NFKC`. It can also be NULL, for no normalization.

If successful, sets `*resultp` to -1 if *s1* < *s2*, 0 if *s1* = *s2*, 1 if *s1* > *s2*, and returns 0. Upon failure, returns -1 with `errno` set.

14.5 Case detection

The following functions determine whether a Unicode string is entirely in upper case. or entirely in lower case, or entirely in title case, or already case-folded.

int u8_is_uppercase (*const uint8_t* ***s**, *size_t* **n**, *const char* ***iso639_language**, *bool* ***resultp**) [Function]
int u16_is_uppercase (*const uint16_t* ***s**, *size_t* **n**, *const char* ***iso639_language**, *bool* ***resultp**) [Function]

int u32_is_uppercase (*const uint32_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]

Sets **resultp* to true if mapping NFD(*s*) to upper case is a no-op, or to false otherwise, and returns 0. Upon failure, returns -1 with errno set.

int u8_is_lowercase (*const uint8_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]
int u16_is_lowercase (*const uint16_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]
int u32_is_lowercase (*const uint32_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]

Sets **resultp* to true if mapping NFD(*s*) to lower case is a no-op, or to false otherwise, and returns 0. Upon failure, returns -1 with errno set.

int u8_is_titlecase (*const uint8_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]
int u16_is_titlecase (*const uint16_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]
int u32_is_titlecase (*const uint32_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]

Sets **resultp* to true if mapping NFD(*s*) to title case is a no-op, or to false otherwise, and returns 0. Upon failure, returns -1 with errno set.

int u8_is_casefolded (*const uint8_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]
int u16_is_casefolded (*const uint16_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]
int u32_is_casefolded (*const uint32_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]

Sets **resultp* to true if applying case folding to NFD(*S*) is a no-op, or to false otherwise, and returns 0. Upon failure, returns -1 with errno set.

The following functions determine whether case mappings have any effect on a Unicode string.

int u8_is_cased (*const uint8_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]
int u16_is_cased (*const uint16_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]
int u32_is_cased (*const uint32_t *s*, *size_t n*, *const char *iso639_language*, *bool *resultp*) [Function]

Sets **resultp* to true if case matters for *s*, that is, if mapping NFD(*s*) to either upper case or lower case or title case is not a no-op. Set **resultp* to false if NFD(*s*) maps to itself under the upper case mapping, under the lower case mapping, and under the title case mapping; in other words, when NFD(*s*) consists entirely of caseless characters. Upon failure, returns -1 with errno set.

15 Regular expressions <uniregex.h>

This include file is not yet implemented.

16 Using the library

This chapter explains some practical considerations, regarding the installation and compiler options that are needed in order to use this library.

16.1 Installation

Before you can use the library, it must be installed. First, you have to make sure all dependencies are installed. They are listed in the file `DEPENDENCIES`.

Then you can proceed to build and install the library, as described in the file `INSTALL`. For installation on Windows systems, please refer to the file `INSTALL.windows`.

16.2 Compiler options

Let's denote as `LIBUNISTRING_PREFIX` the value of the '`--prefix`' option that you passed to `configure` while installing this package. If you didn't pass any '`--prefix`' option, then the package is installed in `/usr/local`.

Let's denote as `LIBUNISTRING_INCLUDEDIR` the directory where the include files were installed. This is usually the same as `${LIBUNISTRING_PREFIX}/include`. Except that if you passed an '`--includedir`' option to `configure`, it is the value of that option.

Let's further denote as `LIBUNISTRING_LIBDIR` the directory where the library itself was installed. This is the value that you passed with the '`--libdir`' option to `configure`, or otherwise the same as `${LIBUNISTRING_PREFIX}/lib`. Recall that when building in 64-bit mode on a 64-bit GNU/Linux system that supports executables in either 64-bit mode or 32-bit mode, you should have used the option `--libdir=${LIBUNISTRING_PREFIX}/lib64`.

So that the compiler finds the include files, you have to pass it the option `-I${LIBUNISTRING_INCLUDEDIR}`.

So that the compiler finds the library during its linking pass, you have to pass it the options `-L${LIBUNISTRING_LIBDIR} -lunistring`. On some systems, in some configurations, you also have to pass options needed for linking with `libiconv`. The autoconf macro `gl_LIBUNISTRING` (see Section 16.4 [Autoconf macro], page 72) deals with this particularity.

16.3 Include files

Most of the include files have been presented in the introduction, see Chapter 1 [Introduction], page 1, and subsequent detailed chapters.

Another include file is `<unistring/version.h>`. It contains the version number of the libunistring library.

`int _LIBUNISTRING_VERSION` [Macro]

> This constant contains the version of libunistring that is being used at compile time. It encodes the major and minor parts of the version number only. These parts are encoded in the form `(major<<8) + minor`.

`int _libunistring_version` [Constant]

> This constant contains the version of libunistring that is being used at run time. It encodes the major and minor parts of the version number only. These parts are encoded in the form `(major<<8) + minor`.

It is possible that _libunistring_version is greater than _LIBUNISTRING_VERSION. This can happen when you use libunistring as a shared library, and a newer, binary backward-compatible version has been installed after your program that uses libunistring was installed.

16.4 Autoconf macro

GNU Gnulib provides an autoconf macro that tests for the availability of libunistring. It is contained in the Gnulib module 'libunistring', see https://www.gnu.org/software/gnulib/MODULES.html#module=libunistring.

The macro is called gl_LIBUNISTRING. It searches for an installed libunistring. If found, it sets and AC_SUBSTs HAVE_LIBUNISTRING=yes and the LIBUNISTRING and LTLIBUNISTRING variables and augments the CPPFLAGS variable, and defines the C macro HAVE_LIBUNISTRING to 1. Otherwise, it sets and AC_SUBSTs HAVE_LIBUNISTRING=no and LIBUNISTRING and LTLIBUNISTRING to empty.

The complexities that gl_LIBUNISTRING deals with are the following:

- On some operating systems, in some configurations, libunistring depends on libiconv, and the options for linking with libiconv must be mentioned explicitly on the link command line.
- GNU libunistring, if installed, is not necessarily already in the search path (CPPFLAGS for the include file search path, LDFLAGS for the library search path).
- GNU libunistring, if installed, is not necessarily already in the run time library search path. To avoid the need for setting an environment variable like LD_LIBRARY_PATH, the macro adds the appropriate run time search path options to the LIBUNISTRING variable. This works on most systems.

16.5 Reporting problems

If you encounter any problem, please don't hesitate to submit a detailed bug report either in the bug tracker at the project page https://savannah.gnu.org/projects/libunistring, or by email to the bug-libunistring@gnu.org mailing list.

Please always include the version number of this library, and a short description of your operating system and compilation environment with corresponding version numbers.

For problems that appear while building and installing libunistring, for which you don't find the remedy in the INSTALL file, please include a description of the options that you passed to the 'configure' script.

17 More advanced functionality

For bidirectional reordering of strings, we recommend the GNU FriBidi library: `http://www.fribidi.org/`.

For the rendering of Unicode strings outside of the context of a given toolkit (KDE/Qt or GNOME/Gtk), we recommend the Pango library: `https://www.pango.org/`.

Appendix A The wchar_t mess

The ISO C and POSIX standard creators made an attempt to fix the first problem mentioned in the section Section 1.5 [char * strings], page 4. They introduced

- a type 'wchar_t', designed to encapsulate an entire character,
- a "wide string" type 'wchar_t *', and
- functions declared in <wctype.h> that were meant to supplant the ones in <ctype.h>.

Unfortunately, this API and its implementation has numerous problems:

- On AIX and Windows platforms, wchar_t is a 16-bit type. This means that it can never accommodate an entire Unicode character. Either the wchar_t * strings are limited to characters in UCS-2 (the "Basic Multilingual Plane" of Unicode), or — if wchar_t * strings are encoded in UTF-16 — a wchar_t represents only half of a character in the worst case, making the <wctype.h> functions pointless.
- On Solaris and FreeBSD, the wchar_t encoding is locale dependent and undocumented. This means, if you want to know any property of a wchar_t character, other than the properties defined by <wctype.h> — such as whether it's a dash, currency symbol, paragraph separator, or similar —, you have to convert it to char * encoding first, by use of the function wctomb.
- When you read a stream of wide characters, through the functions fgetwc and fgetws, and when the input stream/file is not in the expected encoding, you have no way to determine the invalid byte sequence and do some corrective action. If you use these functions, your program becomes "garbage in - more garbage out" or "garbage in - abort".

As a consequence, it is better to use multibyte strings, as explained in the section Section 1.5 [char * strings], page 4. Such multibyte strings can bypass limitations of the wchar_t type, if you use functions defined in gnulib and libunistring for text processing. They can also faithfully transport malformed characters that were present in the input, without requiring the program to produce garbage or abort.

Appendix B The char32_t problem

In response to the wchar_t mess described in the previous section, ISO C 11 introduces two new types: char32_t and char16_t.

char32_t is a type like wchar_t, with the added guarantee that it is 32 bits wide. So, it is a type that is appropriate for encoding a Unicode character. It is meant to resolve the problems of the 16-bit wide wchar_t on AIX and Windows platforms, and allow a saner programming model for wide character strings across all platforms.

char16_t is a type like wchar_t, with the added guarantee that it is 16 bits wide. It is meant to allow porting programs that use the broken wide character strings programming model from Windows to all platforms. Of course, no one needs this.

These types are accompanied with a syntax for defining wide string literals with these element types: u"..." and U"...".

So far, so good. What the ISO C designers forgot, is to provide standardized C library functions that operate on these wide character strings. They standardized only the most basic functions, mbrtoc32 and c32rtomb, which are analogous to mbrtowc and wcrtomb, respectively. For the rest, GNU gnulib https://www.gnu.org/software/gnulib/ provides the functions:

- Functions for converting an entire string: mbstoc32s – like mbstowcs, c32stombs – like wcstombs.
- Functions for testing the properties of a 32-bit wide character: c32isalnum, c32isalpha, etc. – like iswalnum, iswalpha, etc.

Still, this API has two problems:

- The char32_t encoding is locale dependent and undocumented. This means, if you want to know any property of a char32_t character, other than the properties defined by <wctype.h> – such as whether it's a dash, currency symbol, paragraph separator, or similar –, you have to convert it to char * encoding first, by use of the function c32tomb.
- Even on platforms where wchar_t is 32 bits wide, the char32_t encoding may be different from the wchar_t encoding.

Appendix C Licenses

The files of this package are covered by the licenses indicated in each particular file or directory. Here is a summary:

- The `libunistring` library and its header files are dual-licensed under "the GNU LGPLv3+ or the GNU GPLv2+". This means, you can use it under either
  - – the terms of the GNU Lesser General Public License (LGPL) version 3 or (at your option) any later version, or
  - – the terms of the GNU General Public License (GPL) version 2 or (at your option) any later version, or
  - – the same dual license "the GNU LGPLv3+ or the GNU GPLv2+".

  You find the GNU LGPL version 3 in Section C.2 [GNU LGPL], page 88. This license is based on the GNU GPL version 3, see Section C.1 [GNU GPL], page 77.
  You can find the GNU GPL version 2 at `https://www.gnu.org/licenses/old-licenses/gpl-2.0.html`.
  Note: This dual license makes it possible for the `libunistring` library to be used by packages under GPLv2 or GPLv2+ licenses, in particular. See the table in `https://www.gnu.org/licenses/gpl-faq.html#AllCompatibility`.
- This manual is free documentation. It is dually licensed under the GNU FDL and the GNU GPL. This means that you can redistribute this manual under either of these two licenses, at your choice.
  This manual is covered by the GNU FDL. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License (FDL), either version 1.2 of the License, or (at your option) any later version published by the Free Software Foundation (FSF); with no Invariant Sections, with no Front-Cover Text, and with no Back-Cover Texts. A copy of the license is included in Section C.3 [GNU FDL], page 91.
  This manual is covered by the GNU GPL. You can redistribute it and/or modify it under the terms of the GNU General Public License (GPL), either version 3 of the License, or (at your option) any later version published by the Free Software Foundation (FSF). A copy of the license is included in Section C.1 [GNU GPL], page 77.

C.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see https://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `https://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `https://www.gnu.org/licenses/why-not-lgpl.html`.

C.2 GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

   As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

   "The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

   An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

   A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

   The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

   The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

   You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

   If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

   a. under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

   b. under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

   The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

   a. Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

   b. Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

   You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

   a. Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

   b. Accompany the Combined Work with a copy of the GNU GPL and this license document.

   c. For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

   d. Do one of the following:

      0. Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

      1. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

   e. Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

   You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

   a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

   b. Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

   The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

   If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

C.3 GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

L

M

N

O

P

R

S

T

U

V

W