Package: utf8 (via r-universe)

May 24, 2026

Title Unicode Text Processing

Version 1.2.6.9012

Description Process and print 'UTF-8' encoded international text (Unicode). Input, validate, normalize, encode, format, and display.

License Apache License (== 2.0) | file LICENSE

URL https://krlmlr.github.io/utf8/, https://github.com/krlmlr/utf8

BugReports https://github.com/krlmlr/utf8/issues

Depends R (>= 2.10)

Suggests cli, covr, knitr, rlang, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0.9000

Repository https://krlmlr.r-universe.dev

Date/Publication 2026-05-24 01:28:38 UTC

RemoteUrl https://github.com/krlmlr/utf8

RemoteRef HEAD

RemoteSha 008aab4c20382eb5e55c6a8deca85fb0b3840408

Contents

`utf8-package` *The utf8 Package*

Description

UTF-8 Text Processing

Details

Functions for manipulating and printing UTF-8 encoded text:

- `as_utf8()` attempts to convert character data to UTF-8, throwing an error if the data is invalid;
- `utf8_valid()` tests whether character data is valid according to its declared encoding;
- `utf8_normalize()` converts text to Unicode composed normal form (NFC), optionally applying case-folding and compatibility maps;
- `utf8_encode()` encodes a character string, escaping all control characters, so that it can be safely printed to the screen;
- `utf8_format()` formats a character vector by truncating to a specified character width limit or by left, right, or center justifying;
- `utf8_print()` prints UTF-8 character data to the screen;
- `utf8_width()` measures the display width of UTF-8 character strings (many emoji and East Asian characters are twice as wide as other characters);
- `output_ansi()` and `output_utf8()` test for the output connections capabilities.

For a complete list of functions, use `library(help = "utf8")`.

Author(s)

Maintainer: Kirill Müller <kirill@cynkra.com> (ORCID)

Authors:

- Patrick O. Perry [copyright holder]

Other contributors:

- Unicode, Inc. (Unicode Character Database) [copyright holder, data contributor]

See Also

Useful links:

- https://krlmlr.github.io/utf8/
- https://github.com/krlmlr/utf8
- Report bugs at https://github.com/krlmlr/utf8/issues

as_utf8 *UTF-8 Character Encoding*

Description

UTF-8 text encoding and validation

`as_utf8()` converts a character object from its declared encoding to a valid UTF-8 character object, or throws an error if no conversion is possible. If `normalize = TRUE`, then the text gets transformed to Unicode composed normal form (NFC) after conversion to UTF-8.

`utf8_valid()` tests whether the elements of a character object can be translated to valid UTF-8 strings.

Usage

```
as_utf8(x, normalize = FALSE)

utf8_valid(x)
```

Arguments

`x`	character object.
`normalize`	a logical value indicating whether to convert to Unicode composed normal form (NFC).

Value

For `as_utf8()`, the result is a character object with the same attributes as `x` but with `Encoding` set to `"UTF-8"`.

For `utf8_valid()` a logical object with the same `names`, `dim`, and `dimnames` as `x`.

See Also

`utf8_normalize()`, `iconv()`.

Examples

```
# the second element is encoded in latin-1, but declared as UTF-8
x <- c("fa\u00E7ile", "fa\xE7ile", "fa\xC3\xA7ile")
Encoding(x) <- c("UTF-8", "UTF-8", "bytes")

# attempt to convert to UTF-8 (fails)
## Not run: as_utf8(x)

y <- x
Encoding(y[2]) <- "latin1" # mark the correct encoding
as_utf8(y) # succeeds
```

```
# test for valid UTF-8
utf8_valid(x)
```

`output_ansi` *Output Capabilities*

Description

Test whether the output connection has ANSI style escape support or UTF-8 support.

Usage

```
output_ansi()

output_utf8()
```

Details

`output_ansi()` tests whether the output connection supports ANSI style escapes. This is `TRUE` if the connection is a terminal and not the Windows GUI. Otherwise, it is true if running in RStudio 1.1 or later with ANSI escapes enabled, provided `stdout()` has not been redirected to another connection by `sink()`.

`output_utf8()` tests whether the output connection supports UTF-8. For most platforms `l10n_info()$"UTF-8"` gives this information, but this does not give an accurate result for Windows GUIs. To work around this, we proceed as follows:

- if the character locale (`LC_CTYPE`) is `"C"`, then the result is `FALSE`;
- otherwise, if `l10n_info()$"UTF-8"` is `TRUE`, then the result is `TRUE`;
- if running on Windows, then the result is `TRUE`;
- in all other cases the result is `FALSE`.

Strictly speaking, UTF-8 support is always available on Windows GUI, but only a subset of UTF-8 is available (defined by the current character locale) when the output is redirected by `knitr` or another process. Unfortunately, it is impossible to set the character locale to UTF-8 on Windows. Further, the `utf8` package only handles two character locales: C and UTF-8. To get around this, on Windows, we treat all non-C locales on that platform as UTF-8. This liberal approach means that characters in the user's locale never get escaped; others will get output as `<U+XXXX>`, with incorrect values for `utf8_width()`.

Value

A logical scalar indicating whether the output connection supports the given capability.

See Also

`.Platform()`, `isatty()`, `l10n_info()`, `Sys.getlocale()`

Examples

```
# test whether ANSI style escapes or UTF-8 output are supported
cat("ANSI:", output_ansi(), "\n")
cat("UTF8:", output_utf8(), "\n")

# switch to C locale
Sys.setlocale("LC_CTYPE", "C")
cat("ANSI:", output_ansi(), "\n")
cat("UTF8:", output_utf8(), "\n")

# switch to native locale
Sys.setlocale("LC_CTYPE", "")

tmp <- tempfile()
sink(tmp) # redirect output to a file
cat("ANSI:", output_ansi(), "\n")
cat("UTF8:", output_utf8(), "\n")
sink() # restore stdout

# inspect the output
readLines(tmp)
```

---

utf8_encode *Encode Character Object as for UTF-8 Printing*

Description

Escape the strings in a character object, optionally adding quotes or spaces, adjusting the width for display.

Usage

```
utf8_encode(
  x,
  ...,
  width = 0L,
  quote = FALSE,
  justify = "left",
  escapes = NULL,
  display = FALSE,
  utf8 = NULL
)
```

Arguments

x	character object.
...	These dots are for future extensions and must be empty.

width	integer giving the minimum field width; specify NULL or NA for no minimum.
quote	logical scalar indicating whether to surround results with double-quotes and escape internal double-quotes.
justify	justification; one of "left", "right", "centre", or "none". Can be abbreviated.
escapes	a character string specifying the display style for the backslash escapes, as an ANSI SGR parameter string, or NULL for no styling.
display	logical scalar indicating whether to optimize the encoding for display, not byte-for-byte data transmission.
utf8	logical scalar indicating whether to encode for a UTF-8 capable display (ASCII-only otherwise), or NULL to encode for output capabilities as determined by output_utf8().

Details

utf8_encode() encodes a character object for printing on a UTF-8 device by escaping controls characters and other non-printable characters. When display = TRUE, the function optimizes the encoding for display by removing default ignorable characters (soft hyphens, zero-width spaces, etc.) and placing zero-width spaces after wide emoji. When output_utf8() is FALSE the function escapes all non-ASCII characters and gives the same results on all platforms.

Value

A character object with the same attributes as x but with Encoding set to "UTF-8".

See Also

utf8_print().

Examples

```
# the second element is encoded in latin-1, but declared as UTF-8
x <- c("fa\u00E7ile", "fa\xE7ile", "fa\xC3\xA7ile")
Encoding(x) <- c("UTF-8", "UTF-8", "bytes")

# encoding
utf8_encode(x)

# add style to the escapes
cat(utf8_encode("hello\nstyled\\world", escapes = "1"), "\n")
```

utf8_format *UTF-8 Text Formatting*

Description

Format a character object for UTF-8 printing.

Usage

```
utf8_format(
  x,
  ...,
  trim = FALSE,
  chars = NULL,
  justify = "left",
  width = NULL,
  na.encode = TRUE,
  quote = FALSE,
  na.print = NULL,
  print.gap = NULL,
  utf8 = NULL
)
```

Arguments

x	character object.
...	These dots are for future extensions and must be empty.
trim	logical scalar indicating whether to suppress padding spaces around elements.
chars	integer scalar indicating the maximum number of character units to display. Wide characters like emoji take two character units; combining marks and default ignorables take none. Longer strings get truncated and suffixed or prefixed with an ellipsis ("...." or "\u2026", whichever is most appropriate for the current character locale). Set to NULL to limit output to the line width as determined by getOption("width").
justify	justification; one of "left", "right", "centre", or "none". Can be abbreviated.
width	the minimum field width; set to NULL or 0 for no restriction.
na.encode	logical scalar indicating whether to encode NA values as character strings.
quote	logical scalar indicating whether to format for a context with surrounding double-quotes ('"') and escaped internal double-quotes.
na.print	character string (or NULL) indicating the encoding for NA values. Ignored when na.encode is FALSE.
print.gap	non-negative integer (or NULL) giving the number of spaces in gaps between columns; set to NULL or 1 for a single space.

utf8	logical scalar indicating whether to format for a UTF-8 capable display (ASCII-only otherwise), or NULL to format for output capabilities as determined by output_utf8().

Details

utf8_format() formats a character object for printing, optionally truncating long character strings.

Value

A character object with the same attributes as x but with Encoding set to "UTF-8" for elements that can be converted to valid UTF-8 and "bytes" for others.

See Also

utf8_print(), utf8_encode().

Examples

```
# the second element is encoded in latin-1, but declared as UTF-8
x <- c("fa\u00E7ile", "fa\xE7ile", "fa\xC3\xA7ile")
Encoding(x) <- c("UTF-8", "UTF-8", "bytes")

# formatting
utf8_format(x, chars = 3)
utf8_format(x, chars = 3, justify = "centre", width = 10)
utf8_format(x, chars = 3, justify = "right")
```

---

utf8_normalize *Text Normalization*

Description

Transform text to normalized form, optionally mapping to lowercase and applying compatibility maps.

Usage

```
utf8_normalize(
  x,
  ...,
  map_case = FALSE,
  map_compat = FALSE,
  map_quote = FALSE,
  remove_ignorable = FALSE
)
```

Arguments

x	character object.
...	These dots are for future extensions and must be empty.
map_case	a logical value indicating whether to apply Unicode case mapping to the text. For most languages, this transformation changes uppercase characters to their lowercase equivalents.
map_compat	a logical value indicating whether to apply Unicode compatibility mappings to the characters, those required for NFKC and NFKD normal forms.
map_quote	a logical value indicating whether to replace curly single quotes and Unicode apostrophe characters with ASCII apostrophe (U+0027).
remove_ignorable	a logical value indicating whether to remove Unicode "default ignorable" characters like zero-width spaces and soft hyphens.

Details

`utf8_normalize()` converts the elements of a character object to Unicode normalized composed form (NFC) while applying the character maps specified by the `map_case`, `map_compat`, `map_quote`, and `remove_ignorable` arguments.

Value

The result is a character object with the same attributes as `x` but with `Encoding` set to `"UTF-8"`.

See Also

`as_utf8()`.

Examples

```
angstrom <- c("\u00c5", "\u0041\u030a", "\u212b")
utf8_normalize(angstrom) == "\u00c5"
```

---

utf8_print *Print UTF-8 Text*

Description

Print a UTF-8 character object.

Usage

```
utf8_print(
  x,
  ...,
  chars = NULL,
  quote = TRUE,
  na.print = NULL,
  print.gap = NULL,
  right = FALSE,
  max = NULL,
  names = NULL,
  rownames = NULL,
  escapes = NULL,
  display = TRUE,
  style = TRUE,
  utf8 = NULL
)
```

Arguments

x	character object.
...	These dots are for future extensions and must be empty.
chars	integer scalar indicating the maximum number of character units to display. Wide characters like emoji take two character units; combining marks and default ignorables take none. Longer strings get truncated and suffixed or prefixed with an ellipsis ("...") in C locale, "\u2026" in others). Set to NULL to limit output to the line width as determined by getOption("width").
quote	logical scalar indicating whether to put surrounding double-quotes ('"') around character strings and escape internal double-quotes.
na.print	character string (or NULL) indicating the encoding for NA values. Ignored when na.encode is FALSE.
print.gap	non-negative integer (or NULL) giving the number of spaces in gaps between columns; set to NULL or 1 for a single space.
right	logical scalar indicating whether to right-justify character strings.
max	non-negative integer (or NULL) indicating the maximum number of elements to print; set to getOption("max.print") if argument is NULL.
names	a character string specifying the display style for the (column) names, as an ANSI SGR parameter string.
rownames	a character string specifying the display style for the row names, as an ANSI SGR parameter string.
escapes	a character string specifying the display style for the backslash escapes, as an ANSI SGR parameter string.
display	logical scalar indicating whether to optimize the encoding for display, not byte-for-byte data transmission.

style	logical scalar indicating whether to apply ANSI terminal escape codes to style the output. Ignored when output_ansi() is FALSE.
utf8	logical scalar indicating whether to optimize results for a UTF-8 capable display, or NULL to set as the result of output_utf8(). Ignored when output_utf8() is FALSE.

Details

utf8_print() prints a character object after formatting it with utf8_format().

For ANSI terminal output (when output_ansi() is TRUE), you can style the row and column names with the rownames and names parameters, specifying an ANSI SGR parameter string; see https://en.wikipedia.org/wiki/ANSI_escape_code#SGR_(Select_Graphic_Rendition)_parameters.

Value

The function returns x invisibly.

See Also

utf8_format().

Examples

```
# printing (assumes that output is capable of displaying Unicode 10.0.0)
print(intToUtf8(0x1F600 + 0:79)) # with default R print function
utf8_print(intToUtf8(0x1F600 + 0:79)) # with utf8_print, truncates line
utf8_print(intToUtf8(0x1F600 + 0:79), chars = 1000) # higher character limit

# in C locale, output ASCII (same results on all platforms)
oldlocale <- Sys.getlocale("LC_CTYPE")
invisible(Sys.setlocale("LC_CTYPE", "C")) # switch to C locale
utf8_print(intToUtf8(0x1F600 + 0:79))
invisible(Sys.setlocale("LC_CTYPE", oldlocale)) # switch back to old locale

# Mac and Linux only: style the names
# see https://en.wikipedia.org/wiki/ANSI_escape_code#SGR_(Select_Graphic_Rendition)_parameters
utf8_print(matrix(as.character(1:20), 4, 5),
           names = "1;4", rownames = "2;3")
```

---

utf8_width *Measure the Character String Width*

Description

Compute the display widths of the elements of a character object.

Usage

```
utf8_width(x, ..., encode = TRUE, quote = FALSE, utf8 = NULL)
```

Arguments

x	character object.
...	These dots are for future extensions and must be empty.
encode	whether to encode the object before measuring its width.
quote	whether to quote the object before measuring its width.
utf8	logical scalar indicating whether to determine widths assuming a UTF-8 capable display (ASCII-only otherwise), or NULL to format for output capabilities as determined by output_utf8().

Details

utf8_width() returns the printed widths of the elements of a character object on a UTF-8 device (or on an ASCII device when output_utf8() is FALSE), when printed with utf8_print(). If the string is not printable on the device, for example if it contains a control code like "\n", then the result is NA. If encode = TRUE, the default, then the function returns the widths of the encoded elements via utf8_encode()); otherwise, the function returns the widths of the original elements.

Value

An integer object, with the same names, dim, and dimnames as x.

See Also

utf8_print().

Examples

```
# the second element is encoded in latin-1, but declared as UTF-8
x <- c("fa\u00E7ile", "fa\xE7ile", "fa\xC3\xA7ile")
Encoding(x) <- c("UTF-8", "UTF-8", "bytes")

# get widths
utf8_width(x)
utf8_width(x, encode = FALSE)
utf8_width('"')
utf8_width('"', quote = TRUE)
```

Index