Package ‘stringx’

January 11, 2025

Type Package

Title Replacements for Base String Functions Powered by 'stringi'

Version 0.2.9

Date 2025-01-11

Description English is the native language for only 5% of the World population.
Also, only 17% of us can understand this text. Moreover, the Latin alphabet
is the main one for merely 36% of the total.
The early computer era, now a very long time ago, was dominated by the US.
Due to the proliferation of the internet, smartphones, social media, and
other technologies and communication platforms, this is no longer the case.
This package replaces base R string functions (such as grep(),
tolower(), sprintf(), and strptime()) with ones that fully
support the Unicode standards related to natural language and
date-time processing. It also fixes some long-standing inconsistencies,
and introduces some new, useful features.
Thanks to 'ICU' (International Components for Unicode) and 'stringi',
they are fast, reliable, and portable across different platforms.

BugReports `https://github.com/gagolews/stringx/issues`

URL `https://stringx.gagolewski.com/`,
`https://github.com/gagolews/stringx`

License GPL (>= 2)

Depends R (>= 4.1.0)

Imports stringi (>= 1.7.2)

Suggests realtest (>= 0.2.1)

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Marek Gagolewski [aut, cre, cph]
(<`https://orcid.org/0000-0003-0637-6028`>)

Maintainer Marek Gagolewski <`marek@gagolewski.com`>

Repository CRAN

Date/Publication 2025-01-11 17:00:02 UTC

Contents

`about_stringx` *Drop-in Replacements for Base String Functions Powered by Stringi*

Description

stringx reimplements the built-in R string processing functions based on **stringi** – a mature R package for fast, correct, consistent, and convenient text manipulation. Thanks to the **ICU** library, we obtain predictable results on every platform, in each locale, and under any native character encoding.

Keywords: R, text processing, character strings, internationalisation, localisation, ICU, ICU4C, i18n, l10n, Unicode

License: GNU General Public License version 2 or later

Author(s)

Marek Gagolewski

References

stringi *Package homepage*, `https://stringi.gagolewski.com/`

ICU – International Components for Unicode, `https://icu.unicode.org/`

The Unicode Consortium, `https://home.unicode.org/`

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

`digits_dec` *Character Constants*

Description

Letters and digits sets complementing the built-in `LETTERS` and `letters`, see Constants.

Note: calling, e.g., `tolower` on `LETTERS_FRAK` in the current version of **ICU** does not currently yield `letters_frak`.

Usage

```
digits_dec

digits_hex

letters_greek

LETTERS_GREEK

letters_bb

LETTERS_BB

letters_cal

LETTERS_CAL

letters_frak

LETTERS_FRAK

letters_bf

LETTERS_BF
```

Format

Decimal digits

Hexadecimal digits

Greek letters (lower case)

Greek letters (upper case)

Blackboard bold English letters (lower case)

Blackboard bold English letters (upper case)

Calligraphy (script) English letters (lower case)

Calligraphy (script) English letters (upper case)

Fraktur English letters (lower case)

Fraktur English letters (upper case)

Bold English letters (lower case)

Bold English letters (upper case)

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at `https://stringx.gagolewski.com/`

Examples

```
letters_bb
letters_bf
letters_cal
letters_frak
letters_greek
LETTERS_BB
LETTERS_BF
LETTERS_CAL
LETTERS_FRAK
LETTERS_GREEK
digits_dec
digits_hex
```

`grepl2` *Detect Pattern Occurrences*

Description

`grepl2` indicates whether a string matches the corresponding pattern or not.

`grepv2` returns a subset of `x` matching the corresponding patterns. Its replacement version allows for substituting such a subset with new content.

Usage

```
grepl2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE, invert = FALSE)

grepv2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE, invert = FALSE)

grepv2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE, invert = FALSE) <- value

grepl(
  pattern,
  x,
  ...,
  ignore.case = FALSE,
  fixed = FALSE,
  invert = FALSE,
  perl = FALSE,
  useBytes = FALSE
)

grep(
  pattern,
  x,
  ...,
  ignore.case = FALSE,
  fixed = FALSE,
  value = FALSE,
  invert = FALSE,
  perl = FALSE,
  useBytes = FALSE
)
```

Arguments

`x` character vector whose elements are to be examined

`pattern` character vector of nonempty search patterns; for `grepv2` and `grep`, must not be longer than `x`

`...` further arguments to `stri_detect`, e.g., `max_count`, `locale`, `dotall`

`ignore_case`, `ignore.case`
single logical value; indicates whether matching should be case-insensitive

`fixed` single logical value; `FALSE` for matching with regular expressions (see about_search_regex); `TRUE` for fixed pattern matching (about_search_fixed); `NA` for the Unicode collation algorithm (about_search_coll)

`invert` single logical value; indicates whether a no-match is rather of interest

`value` character vector of replacement strings or a single logical value indicating whether indexes of strings in `x` matching patterns should be returned

`perl`, `useBytes` not used (with a warning if attempting to do so) [DEPRECATED]

Details

These functions are fully vectorised with respect to x and pattern.

The [DEPRECATED] grepl simply calls grepl2 which have a cleaned-up argument list.

The [DEPRECATED] grep with value=FALSE is actually redundant – it can be trivially reproduced with grepl and which.

grepv2 and grep with value=FALSE combine pattern matching and subsetting and some users may find it convenient in conjunction with the forward pipe operator, |>.

Value

grepl2 and [DEPRECATED] grep return a logical vector. They preserve the attributes of the longest inputs (unless they are dropped due to coercion). Missing values in the inputs are propagated consistently.

grepv2 and [DEPRECATED] grep with value=TRUE returns a subset of x with elements matching the corresponding patterns. [DEPRECATED] grep with value=FALSE returns the indexes in x where a match occurred. Missing values are not included in the outputs and only the names attribute is preserved, because the length of the result may be different than that of x.

The replacement version of grepv2 modifies x 'in-place'.

Differences from Base R

grepl and grep are [DEPRECATED] replacements for base grep and grepl implemented with stri_detect.

- there are inconsistencies between the argument order and naming in grepl, strsplit, and startsWith (amongst others); e.g., where the needle can precede the haystack, the use of the forward pipe operator, |>, is less convenient **[fixed by introducing grepl2]**
- base R implementation is not portable as it is based on the system PCRE or TRE library (e.g., some Unicode classes may not be available or matching thereof can depend on the current LC_CTYPE category **[fixed here]**
- not suitable for natural language processing **[fixed here – use fixed=NA]**
- two different regular expression libraries are used (and historically, ERE was used in place of TRE) **[here, ICU Java-like regular expression engine is only available, hence the perl argument has no meaning]**
- not vectorised w.r.t. pattern **[fixed here, however, in grep, pattern cannot be longer than x]**
- missing values in haystack will result in a no-match **[fixed in grepl; see Value]**
- ignore.case=TRUE cannot be used with fixed=TRUE **[fixed here]**
- no attributes are preserved **[fixed here; see Value]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): paste, nchar, strsplit, gsub2, gregexpr2, gregextr2, gsubstr

Examples

```
x <- c("abc", "1237", "\U0001f602", "\U0001f603", "stringx\U0001f970", NA)
grepl2(x, "\\p{L}")
which(grepl2(x, "\\p{L}"))  # like grep

# at least 1 letter or digit:
p <- c("\\p{L}", "\\p{N}")
`dimnames<-`(outer(x, p, grepl2), list(x, p))

x |> grepv2("\\p{L}")
grepv2(x, "\\p{L}", invert=TRUE) <- "\U0001F496"
print(x)
```

ISOdatetime *Construct Date-time Objects*

Description

ISOdate and ISOdatetime construct date-time objects from numeric representations. Sys.time returns current time.

Usage

```
ISOdatetime(
  year,
  month,
  day,
  hour,
  min,
  sec,
  tz = "",
  lenient = FALSE,
  locale = NULL
)

ISOdate(
  year,
  month,
  day,
  hour = 0L,
  min = 0L,
  sec = 0L,
  tz = "",
  lenient = FALSE,
  locale = NULL
)

Sys.time()
```

Arguments

`year, month, day, hour, min, sec`
numeric vectors

`tz` `NULL` or `''` for the default time zone (see `stri_timezone_get`) or a single string with a timezone identifier, see `stri_timezone_list`

`lenient` single logical value; should date/time parsing be lenient?

`locale` `NULL` or `''` for the default locale (see `stri_locale_get`) or a single string with a locale identifier, see `stri_locale_list`

Value

These functions return an object of class `POSIXxt`, which extends upon `POSIXct`, `strptime`.

You might wish to consider calling `as.Date` on the result yielded by `ISOdate`.

No attributes are preserved (because they are too many).

Differences from Base R

Replacements for base `ISOdatetime` and `ISOdate` implemented with `stri_datetime_create`.

- `ISOdate` does not treat dates as being at midnight by default **[fixed here]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at `https://stringx.gagolewski.com/`

Related function(s): `strptime`

Examples

```
ISOdate(1970, 1, 1)
ISOdatetime(1970, 1, 1, 12, 0, 0)
```

nchar *Get Length or Width of Strings*

Description

`nchar` computes the number of code points, bytes used, or estimated total width of strings in a character vector. `nzchar` indicates which strings are empty.

Usage

```
nchar(x, type = "chars", allowNA = FALSE, keepNA = TRUE)

nzchar(x, keepNA = TRUE)
```

Arguments

`x`	character vector or an object coercible to
`type`	`"chars"` gives the number of code points, `"width"` estimates the string width, `"bytes"` computes the number of bytes
`allowNA`	not used (with a warning if attempting to do so) [DEPRECATED]
`keepNA`	if `FALSE`, missing values will be treated as `"NA"` strings; otherwise, the corresponding outputs will be missing as well [DEPRECATED]

Details

String width might be useful when displaying text using a monospaced font.

Value

`nchar` returns an integer vector.

`nzchar` returns a logical vector, where `TRUE` indicates that the corresponding string is of non-zero length (i.e., non-empty).

Differences from Base R

Replacement for base `nchar` and `nzchar` implemented with `stri_length`, `stri_width`, `stri_numbytes`, and `stri_isempty`.

- `keepNA` does not default to `TRUE`, and hence missing values are treated as `"NA"` strings **[fixed here]**
- some emojis, combining characters and modifiers (e.g., skin tones) are not recognised properly **[fixed here]**
- only the `names` attribute is propagated **[fixed here]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): sprintf, substr, strtrim

Examples

```
x <- c(
    "\U0001F4A9",
    "\U0001F64D\U0001F3FC\U0000200D\U00002642\U0000FE0F",
    "\U0001F64D\U0001F3FB\U0000200D\U00002642",
    "\U000026F9\U0001F3FF\U0000200D\U00002640\U0000FE0F",
    "\U0001F3F4\U000E0067\U000E0062\U000E0073\U000E0063\U000E0074\U000E007F"
)
print(x)
base::nchar(x, "width")
stringx::nchar(x, "width")
```

paste *Concatenate Strings*

Description

Concatenate (join) the corresponding and/or consecutive elements of given vectors, after converting them to strings.

Usage

```
paste(..., sep = " ", collapse = NULL, recycle0 = FALSE)

paste0(..., sep = "", collapse = NULL, recycle0 = FALSE)

e1 %x+% e2

strcat(x, collapse = "", na.rm = FALSE)
```

Arguments

`...` character vectors (or objects coercible to) whose corresponding/consecutive elements are to be concatenated

`sep` single string; separates terms

`collapse` single string or `NULL`; an optional separator if tokens are to be merged into a single string

`recycle0` single logical value; if `FALSE`, then empty vectors provided via `...` are silently ignored e1, e2 character vectors (or objects coercible to) whose corresponding elements are to be concatenated

x character vector (or an object coercible to) whose consecutive elements are to be concatenated

na.rm single logical value; if TRUE, missing values are silently ignored

Details

`%x+%` is an operator that concatenates corresponding strings from two character vectors (and which behaves just like the arithmetic `+` operator).

strcat joins (aggregates based on string concatenation) consecutive strings in a character vector, possibly with a specified separator in place, into a single string.

paste and paste0, concatenate a number of vectors using the same separator and then possibly join them into a single string. We recommend using `%x+%`, sprintf, and strcat instead (see below for discussion).

Value

A character vector (in UTF-8).

`%x+%` preserves object attributes in a similar way as other Arithmetic operators (however, they may be lost during as.character(...) conversion, which is an S3 generic).

strcat is an aggregation function, therefore it preserves no attributes whatsoever.

Currently, paste and paste0 preserve no attributes too.

Differences from Base R

Replacement for base paste implemented with stri_join.

Note that paste can be thought of as a string counterpart of both the `+` operator (actually, some languages do have a binary operator for string concatenation, e.g., `.` in Perl and PHP, `+` (str.__add__) in Python; R should have it too, but does not) which is additionally vectorised ('Map') and the sum function ('Reduce'). Therefore, we would expect it to behave similarly with regards to the propagation of missing values and the preservation of object attributes, but it does not.

- missing values treated as "NA" strings (it is a well-documented feature though) **[fixed here]**
- partial recycling with no warning "longer object length is not a multiple of shorter object length" **[fixed here]**
- empty vectors are treated as vectors of empty strings **[fixed here]**
- input objects' attributes are not preserved **[fixed only in ‘%x+%‘ operator]**
- paste0 multiplies entities without necessity; sep="" should be the default in paste **[not fixed]**
- paste0 treats the named argument sep="..." as one more vector to concatenate **[fixed by introducing sep argument]**

- overloading `` `+.character` `` has no effect in R, because S3 method dispatch is done internally with hard-coded support for character arguments. We could have replaced the generic `` `+` `` with the one that calls `UseMethod`, but the dispatch would be done on the type of the first argument anyway (not to mention it feels like a too intrusive solution). Actually having a separate operator for concatenation (similar to PHP's or Perl's `` `.` ``) which always coerces to character frees the user from manual coercion (is it such a burden on the other hand?) **[fixed by introducing ‘`%x+%`‘ operator]**

It should also be noted that `paste` with `collapse=NULL` is a special case of `sprintf` (which is featured in many programming languages; R's version is of course vectorised). For instance, `paste(x, y, sep=",")` is equivalent to `sprintf("%s,%s", x, y)`.

Taking into account the above, `paste` and `paste0` seem redundant and hence we mark them as [DEPRECATED]. Here are our recommendations:

- the most frequent use case - concatenating corresponding strings from two character vectors with no separator - is covered by a new operator `` `%x+%` `` which propagates NAs correctly and handles object attributes the same way as the built-in arithmetic operators;
- for fancy elementwise (like 'Map') concatenation, use our version of `sprintf`;
- for the 'flattening' of consecutive strings in a character vector (like 'Reduce'), use the new function `strcat`.

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at `https://stringx.gagolewski.com/`

Related function(s): `strrep`, `sprintf`

Examples

```
# behaviour of `+` vs. base::paste vs. stringx::paste
x <- structure(c(x=1, y=NA, z=100, w=1000), F="*")
y1 <- structure(c(a=1, b=2, c=3), G="#", F="@")
y2 <- structure(c(a=1, b=2, c=3, d=4), G="#", F="@")
y3 <- structure(1:4, G="#", F="@", dim=c(2, 2), dimnames=list(NULL, c("a", "b")))
x + y1
x + y2
x + y3
y2 + x
base::paste(x, y1)
base::paste(x, y2)
base::paste(x, y3)
stringx::paste(x, y1)
stringx::paste(x, y2)
stringx::paste(x, y3)
base::paste(x, character(0), y2, sep=",")
stringx::paste(x, character(0), y2, sep=",")
x %x+% y1
```

```
x %x+% y2
x %x+% y3
y2 %x+% x
x %x+% character(0)
strcat(x, collapse=",")
strcat(x, collapse=",", na.rm=TRUE)
```

regexpr2 *Locate Pattern Occurrences*

Description

regexpr2 and gregexpr2 locate, respectively, first and all (i.e., **g**lobally) occurrences of a pattern. regexec2 and gregexec2 can additionally pinpoint the matches to parenthesised subexpressions (regex capture groups).

Usage

```
regexpr2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE)

gregexpr2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE)

regexec2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE)

gregexec2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE)

regexpr(
  pattern,
  x = text,
  ...,
  ignore.case = FALSE,
  fixed = FALSE,
  perl = FALSE,
  useBytes = FALSE,
  text
)

gregexpr(
  pattern,
  x = text,
  ...,
  ignore.case = FALSE,
  fixed = FALSE,
  perl = FALSE,
  useBytes = FALSE,
  text
)

regexec(
  pattern,
  x = text,
  ...,
  ignore.case = FALSE,
  fixed = FALSE,
  perl = FALSE,
  useBytes = FALSE,
  text
)

gregexec(
  pattern,
  x = text,
  ...,
  ignore.case = FALSE,
  fixed = FALSE,
  perl = FALSE,
  useBytes = FALSE,
  text
)
```

Arguments

x
: character vector whose elements are to be examined

pattern
: character vector of nonempty search patterns

...
: further arguments to `stri_locate`, e.g., `omit_empty`, `locale`, `dotall`

ignore_case, ignore.case
: single logical value; indicates whether matching should be case-insensitive

fixed
: single logical value; `FALSE` for matching with regular expressions (see about_search_regex); `TRUE` for fixed pattern matching (about_search_fixed); `NA` for the Unicode collation algorithm (about_search_coll)

perl, useBytes
: not used (with a warning if attempting to do so) [DEPRECATED]

text
: alias to the `x` argument [DEPRECATED]

Details

These functions are fully vectorised with respect to both `x` and `pattern`.

Use `substrl` and `gsubstrl` to extract or replace the identified chunks. Also, consider using `regextr2` and `gregextr2` directly instead.

Value

`regexpr2` and [DEPRECATED] `regexpr` return an integer vector which gives the start positions of the first substrings matching a pattern. The `match.length` attribute gives the corresponding match lengths. If there is no match, the two values are set to `-1`.

gregexpr2 and [DEPRECATED] gregexpr yield a list whose elements are integer vectors with match.length attributes, giving the positions of all the matches. For consistency with regexpr2, a no-match is denoted with a single -1, hence the output is guaranteed to consist of non-empty integer vectors.

regexec2 and [DEPRECATED] regexec return a list of integer vectors giving the positions of the first matches and the locations of matches to the consecutive parenthesised subexpressions (which can only be recognised if fixed=FALSE). Each vector is equipped with the match.length attribute.

gregexec2 and [DEPRECATED] gregexec generate a list of matrices, where each column corresponds to a separate match; the first row is the start index of the match, the second row gives the position of the first captured group, and so forth. Their match.length attributes are matrices of corresponding sizes.

These functions preserve the attributes of the longest inputs (unless they are dropped due to coercion). Missing values in the inputs are propagated consistently.

Differences from Base R

Replacements for base gregexpr (and others) implemented with stri_locate.

- there are inconsistencies between the argument order and naming in grepl, strsplit, and startsWith (amongst others); e.g., where the needle can precede the haystack, the use of the forward pipe operator, |>, is less convenient **[fixed here]**
- base R implementation is not portable as it is based on the system PCRE or TRE library (e.g., some Unicode classes may not be available or matching thereof can depend on the current LC_CTYPE category **[fixed here]**
- not suitable for natural language processing **[fixed here – use** fixed=NA**]**
- two different regular expression libraries are used (and historically, ERE was used in place of TRE) **[here, ICU Java-like regular expression engine is only available, hence the** perl **argument has no meaning]**
- not vectorised w.r.t. pattern **[fixed here]**
- ignore.case=TRUE cannot be used with fixed=TRUE **[fixed here]**
- no attributes are preserved **[fixed here; see Value]**
- in regexec, match.length attribute is unnamed even if the capture groups are (but gregexec sets dimnames of both start positions and lengths) **[fixed here]**
- regexec and gregexec with fixed other than FALSE make little sense. **[this argument is [DEPRECATED] in** regexec2 **and** gregexec2**]**
- gregexec does not always yield a list of matrices **[fixed here]**
- a no-match to a conditional capture group is assigned length 0 **[fixed here]**
- no-matches result in a single -1, even if capture groups are defined in the pattern **[fixed here]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): paste, nchar, strsplit, gsub2, grepl2, gregextr2, gsubstrl

Examples

```
x <- c(aca1="acacaca", aca2="gaca", noaca="actgggca", na=NA)
regexpr2(x, "(A)[ACTG]\\1", ignore_case=TRUE)
regexpr2(x, "aca") >= 0  # like grepl2
gregexpr2(x, "aca", fixed=TRUE, overlap=TRUE)

# two named capture groups:
regexec2(x, "(?<x>a)(?<y>cac?)")
gregexec2(x, "(?<x>a)(?<y>cac?)")

# extraction:
gsubstrl(x, gregexpr2(x, "(A)[ACTG]\\1", ignore_case=TRUE))
gregextr2(x, "(A)[ACTG]\\1", ignore_case=TRUE)  # equivalent
```

regextr2 *Extract Pattern Occurrences*

Description

regextr2 and gregextr2 extract, respectively, first and all (i.e., **g**lobally) occurrences of a pattern. Their replacement versions substitute the matching substrings with new content.

Usage

```
regextr2(
  x,
  pattern,
  ...,
  ignore_case = FALSE,
  fixed = FALSE,
  capture_groups = FALSE
)

gregextr2(
  x,
  pattern,
  ...,
  ignore_case = FALSE,
  fixed = FALSE,
  capture_groups = FALSE
)
```

```
regextr2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE) <- value

gregextr2(x, pattern, ..., ignore_case = FALSE, fixed = FALSE) <- value
```

Arguments

`x` character vector whose elements are to be examined

`pattern` character vector of nonempty search patterns

`...` further arguments to `stri_locate`, e.g., `omit_empty`, `locale`, `dotall`

`ignore_case` single logical value; indicates whether matching should be case-insensitive

`fixed` single logical value; `FALSE` for matching with regular expressions (see about_search_regex); `TRUE` for fixed pattern matching (about_search_fixed); `NA` for the Unicode collation algorithm (about_search_coll)

`capture_groups` single logical value; whether matches individual capture groups should be extracted separately

`value` character vector (for `regextr`) or list of character vectors (for `gregextr`) defining the replacement strings

Details

Convenience functions based on `gregexpr2` and `gsubstrl` (amongst others). Provided as pipe operator-friendly alternatives to [DEPRECATED] `regmatches` and [DEPRECATED] `strcapture`.

They are fully vectorised with respect to `x`, `pattern`, and `value`.

Note that, unlike in `gsub2`, each substituted chunk can be replaced with different content. However, references to matches to capture groups cannot be made.

Value

`capture_groups` is `FALSE`, `regextr2` returns a character vector and `gregextr2` gives a list of character vectors.

Otherwise, `regextr2` returns a list of character vectors, giving the whole match as well as matches to the individual capture groups. In `gregextr2`, this will be a matrix with as many columns as there are matches.

Missing values in the inputs are propagated consistently. In `regextr2`, a no-match is always denoted with `NA` (or series thereof). In `gregextr2`, the corresponding result is empty (unless we mean a no-match to an optional capture group within a matching substring). Note that this function distinguishes between a missing input and a no-match.

Their replacement versions return a character vector.

These functions preserve the attributes of the longest inputs (unless they are dropped due to coercion).

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): paste, nchar, strsplit, gsub2 grepl2, gregexpr2, gsubstrl,

Examples

```
x <- c(aca1="acacaca", aca2="gaca", noaca="actgggca", na=NA)
regextr2(x, "(?<x>a)(?<y>cac?)")
gregextr2(x, "(?<x>a)(?<y>cac?)")
regextr2(x, "(?<x>a)(?<y>cac?)", capture_groups=TRUE)
gregextr2(x, "(?<x>a)(?<y>cac?)", capture_groups=TRUE)

# substitution - note the different replacement strings:
`gregextr2<-`(x, "(?<x>a)(?<y>cac?)", value=list(c("!", "?"), "#"))
# references to capture groups can only be used in gsub and sub:
gsub2(x, "(?<x>a)(?<y>cac?)", "{$1}{$2}")

regextr2(x, "(?<x>a)(?<y>cac?)") <- "\U0001D554\U0001F4A9"
print(x)  # x was modified 'in-place'
```

sprintf *Format Strings*

Description

`sprintf` creates strings from a given template and the arguments provided. A new function (present in C and many other languages), `printf`, displays formatted strings.

Usage

```
sprintf(fmt, ..., na_string = NA_character_)

printf(fmt, ..., file = "", sep = "\n", append = FALSE, na_string = "NA")
```

Arguments

`fmt`	character vector of format strings
`...`	vectors with data to format (coercible to integer, real, or character)
`na_string`	single string to represent missing values; if `NA`, missing values in `...` result in the corresponding outputs be missing too
`file`	see `cat`
`sep`	see `cat`
`append`	see `cat`

Details

Note that the purpose of `printf` is to display a string, not to create a new one for use elsewhere, therefore this function, as an exception, treats missing values as `"NA"` strings.

Value

`sprintf` returns a character vector (in UTF-8). No attributes are preserved. `printf` returns 'nothing'.

Differences from Base R

Replacement for base `sprintf` implemented with `stri_sprintf`.

- missing values in `...` are treated as `"NA"` strings **[fixed in `sprintf`, left in `printf`, but see the `na_string` argument]**
- partial recycling results in an error **[fixed here – warning given]**
- input objects' attributes are not preserved **[not fixed, somewhat tricky]**
- in to-string conversions, field widths and precisions are interpreted as bytes which is of course problematic for text in UTF-8 **[fixed by interpreting these as Unicode code point widths]**
- `fmt` is limited to 8192 bytes and the number of arguments passed via `...` to 99 (note that we can easily exceed this limit by using `do.call`) **[rewritten from scratch, there is no limit anymore]**
- unused values in `...` are evaluated anyway (should not evaluation be lazy?) **[not fixed here because this is somewhat questionable; in both base R and our case, a warning is given if this is the case; moreover, the length of the longest argument always determines the length of the output]**
- coercion of each argument can only be done once **[fixed here - can coerce to integer, real, and character]**
- either width or precision can be fetched from `...`, but not both **[fixed here - two asterisks are allowed in format specifiers]**
- `NA`/`NaN`s are not prefixed by a sign/space even if we explicitly request this **[fixed here - prefixed by a space]**
- the outputs are implementation-dependent; the format strings are passed down to the system (`libc`) `sprintf` function **[not fixed here (yet), but the format specifiers are normalised more eagerly]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at `https://stringx.gagolewski.com/`

Related function(s): `paste`, `strrep`, `strtrim`, `substr`, `nchar`, `strwrap`

Examples

```
# UTF-8 number of bytes vs Unicode code point width:
l <- c("e", "e\u00b2", "\u03c0", "\u03c0\u00b2", "\U0001f602\U0001f603")
r <- c(exp(1), exp(2), pi, pi^2, NaN)
cat(base::sprintf("%8s=%+.3f", l, r), sep="\n")
cat(stringx::sprintf("%8s=%+.3f", l, r), sep="\n")
```

`startsWith` *Detect Pattern Occurrences at Start or End of Strings*

Description

Determines if a string starts or ends with a match to a specified fixed pattern.

Usage

```
startsWith(
  x,
  pattern = prefix,
  ...,
  ignore_case = ignore.case,
  fixed = TRUE,
  ignore.case = FALSE,
  prefix
)

endsWith(
  x,
  pattern = suffix,
  ...,
  ignore_case = ignore.case,
  fixed = TRUE,
  ignore.case = FALSE,
  suffix
)
```

Arguments

`x`	character vector whose elements are to be examined
`pattern`	character vector with patterns to search for
`...`	further arguments to `stri_startswith` and `stri_endswith`, e.g., `locale`
`ignore_case`	single logical value; indicates whether matching should be case-insensitive
`fixed`	single logical value; `TRUE` for fixed pattern matching (see about_search_fixed); `NA` for the Unicode collation algorithm (about_search_coll); `FALSE` is not supported – use `grepl` instead

`ignore.case`	alias to the `ignore_case` argument [DEPRECATED]
`prefix, suffix`	aliases to the `pattern` argument [DEPRECATED]

Details

These functions are fully vectorised with respect to both arguments.

For matching with regular expressions, see `grepl` with patterns like `"^prefix"` and `"suffix$"`.

Value

Each function returns a logical vector, indicating whether a pattern match has been detected or not. They preserve the attributes of the longest inputs (unless they are dropped due to coercion).

Differences from Base R

Replacements for base `startsWith` and `endsWith` implemented with `stri_startswith` and `stri_endswith`.

- there are inconsistencies between the argument order and naming in `grepl`, `strsplit`, and `startsWith` (amongst others); e.g., where the needle can precede the haystack, the use of the forward pipe operator, `|>`, is less convenient **[fixed here]**
- `grepl` also features the `ignore.case` argument **[added here]**
- partial recycling without the usual warning **[fixed here]**
- no attributes preserved whatsoever **[fixed here]**
- not suitable for natural language processing **[fixed here – use `fixed=NA`]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at `https://stringx.gagolewski.com/`

Related function(s): `grepl`, `substr`

Examples

```
startsWith("ababa", c("a", "ab", "aba", "baba", NA))
outer(
    c("aba", "abb", "abc", "baba", "bac"),
    c("A", "B", "C"),
    endsWith,
    ignore_case=TRUE
)
x <- c("Mario", "mario", "M\u00E1rio", "M\u00C1RIO", "Mar\u00EDa", "Rosario")
x[startsWith(x, "mario", ignore_case=TRUE)]
x[startsWith(x, "mario", fixed=NA, strength=1L)]
```

strcoll *Compare Strings*

Description

These functions provide means to compare strings in any locale using the Unicode collation algorithm.

Usage

```
strcoll(
  e1,
  e2,
  locale = NULL,
  strength = 3L,
  alternate_shifted = FALSE,
  french = FALSE,
  uppercase_first = NA,
  case_level = FALSE,
  normalisation = FALSE,
  numeric = FALSE
)

e1 %x<% e2

e1 %x<=% e2

e1 %x==% e2

e1 %x!=% e2

e1 %x>% e2

e1 %x>=% e2
```

Arguments

e1, e2	character vector whose corresponding elements are to be compared
locale	NULL or "" for the default locale (see stri_locale_get) or a single string with a locale identifier, see stri_locale_list
strength	see stri_opts_collator
alternate_shifted	see stri_opts_collator
french	see stri_opts_collator
uppercase_first	see stri_opts_collator

case_level	see stri_opts_collator
normalisation	see stri_opts_collator
numeric	see stri_opts_collator

Details

These functions are fully vectorised with respect to both arguments.

For a locale-insensitive behaviour like that of strcmp from the standard C library, call strcoll(e1, e2, locale="C", strength=4L, normalisation=FALSE). However, some normalisation will still be performed.

Value

strcoll returns an integer vector representing the comparison results: if a string in e1 is smaller than the corresponding string in e2, the corresponding result will be equal to -1, and 0 if they are canonically equivalent, as well as 1 if the former is greater than the latter.

The binary operators call strcoll with default arguments and return logical vectors.

Differences from Base R

Replacements for base Comparison operators implemented with stri_cmp.

- collation in different locales is difficult and non-portable across platforms **[fixed here – using services provided by ICU]**
- overloading `` `<.character` `` has no effect in R, because S3 method dispatch is done internally with hard-coded support for character arguments. We could have replaced the generic `` `<` `` with the one that calls UseMethod, but it feels like a too intrusive solution **[fixed by introducing the ‘%x<%‘ operator]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): xtfrm

Examples

```
# lexicographic vs. numeric sort
strcoll("100", c("1", "10", "11", "99", "100", "101", "1000"))
strcoll("100", c("1", "10", "11", "99", "100", "101", "1000"), numeric=TRUE)
strcoll("hladn\u00FD", "chladn\u00FD", locale="sk_SK")
```

strptime *Parse and Format Date-time Objects*

Description

Note that the date-time processing functions in **stringx** are a work in progress. Feature requests/comments/remarks are welcome.

`strptime` parses strings representing date-time data and converts it to a date-time object.

`strftime` formats a date-time object and outputs it as a character vector.

The functions are meant to be compatible with each other, especially with regards to formatting/printing. This is why they return/deal with objects of a new class, `POSIXxt`, which expends upon the built-in `POSIXct`.

Usage

```
strptime(x, format, tz = "", lenient = FALSE, locale = NULL)

strftime(
  x,
  format = "%Y-%m-%dT%H:%M:%S%z",
  tz = attr(x, "tzone")[1L],
  usetz = FALSE,
  ...,
  locale = NULL
)

## S3 method for class 'POSIXxt'
format(
  x,
  format = "%Y-%m-%dT%H:%M:%S%z",
  tz = attr(x, "tzone")[1L],
  usetz = FALSE,
  ...,
  locale = NULL
)

is.POSIXxt(x)

as.POSIXxt(x, tz = "", ...)

## S3 method for class 'POSIXt'
as.POSIXxt(x, tz = attr(x, "tzone")[1L], ...)

## S3 method for class 'POSIXxt'
as.POSIXlt(x, tz = attr(x, "tzone")[1L], ..., locale = NULL)
```

```
## Default S3 method:
as.POSIXxt(x, tz = "", ...)

## S3 method for class 'POSIXxt'
as.Date(x, ...)

## S3 method for class 'Date'
as.POSIXxt(x, ...)

## S3 method for class 'character'
as.POSIXxt(x, tz = "", format = NULL, ..., lenient = FALSE, locale = NULL)

## S3 method for class 'POSIXxt'
Ops(e1, e2)

## S3 method for class 'POSIXxt'
seq(from, to, by, length.out = NULL, along.with = NULL, ...)

## S3 method for class 'POSIXxt'
c(..., recursive = FALSE)

## S3 method for class 'POSIXxt'
rep(..., recursive = FALSE)
```

Arguments

`x` object to be converted: a character vector for `strptime` and `as.POSIXxt.character`, an object of class `POSIXxt` for `strftime` an object of class `Date` for `as.POSIXxt.Date`, or objects coercible to

`format` character vector of date-time format specifiers, see `stri_datetime_fstr`; e.g., `"%Y-%m-%d"` or `"datetime_full"`; the default conforms to the ISO 8601 guideline

`tz` `NULL` or `''` for the default time zone (see `stri_timezone_get`) or a single string with a timezone identifier, see `stri_timezone_list`; note that when `x` is equipped with `tzone` attribute, this datum is used; `as.POSIXxt.character` treats dates as being at midnight local time

`lenient` single logical value; should date/time parsing be lenient?

`locale` `NULL` or `''` for the default locale (see `stri_locale_get`) or a single string with a locale identifier, see `stri_locale_list`

`usetz` not used (with a warning if attempting to do so) [DEPRECATED]

`...` not used

`e1`, `e2`, `from`, `to`, `by`, `length.out`, `along.with`, `recursive`
arguments to `c`, `rep`, `seq`, etc.

Details

Note that the ISO 8601 guideline suggests a year-month-day date format and a 24-hour time format always indicating the effective time zone, e.g., `2015-12-31T23:59:59+0100`. This is so as to avoid ambiguity.

When parsing strings, missing fields are filled based on today's midnight data.

Value

`strftime` and `format` return a character vector (in UTF-8).

`strptime`, `as.POSIXxt.Date`, and `asPOSIXxt.character` return an object of class `POSIXxt`, which extends upon `POSIXct`, see also DateTimeClasses.

Subtraction returns an object of the class `difftime`, see `difftime`.

If a string cannot be recognised as valid date/time specifier (as per the given format string), the corresponding output will be `NA`.

Differences from Base R

Replacements for base `strptime` and `strftime` implemented with `stri_datetime_parse` and `stri_datetime_format`.

`format.POSIXxt` is a thin wrapper around `strftime`.

- formatting/parsing date-time in different locales and calendars is difficult and non-portable across platforms **[fixed here – using services provided by ICU]**
- default format not conforming to ISO 8601, in particular not displaying the current time zone **[fixed here]**
- only the names attribute in `x` is propagated **[fixed here]**
- partial recycling with no warning **[fixed here]**
- `strptime` returns an object of class `POSIXlt`, which is not the most convenient to work with, e.g., when including in data frames **[fixed here]**
- Ideally, there should be only one class to represent dates and one to represent date/time; `POSIXlt` is no longer needed as we have `stri_datetime_fields`; our new `POSIXxt` class aims to solve the underlying problems with `POSIXct`'s not being consistent with regards to working in different time zones and dates (see, e.g., `as.Date(as.POSIXct(strftime(Sys.Date())))`) **[addressed here]**
- dates without times are not always treated as being at midnight (despite that being stated in the help page for `as.POSIXct`) **[fixed here]**
- `strftime` does not honour the `tzone` attribute, which is used whilst displaying time (via `format`) **[fixed here]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at `https://stringx.gagolewski.com/`

Related function(s): `sprintf`, `ISOdatetime`

Examples

```
strftime(Sys.time())  # default format - ISO 8601
f <- c("date_full", "%Y-%m-%d", "date_relative_short", "datetime_full")
strftime(Sys.time(), f)  # current default locale
strftime(Sys.time(), f, locale="de_DE")
strftime(Sys.time(), "date_short", locale="en_IL@calendar=hebrew")
strptime("1970-01-01 00:00:00", "%Y-%m-%d %H:%M:%S", tz="GMT")
strptime("14 Nisan 5703", "date_short", locale="en_IL@calendar=hebrew")
as.POSIXxt("1970-01-01")
as.POSIXxt("1970/01/01 12:00")
```

`strrep` *Duplicate Strings*

Description

Concatenate a number of copies of each string.

Usage

```
strrep(x, times)

e1 %x*% e2
```

Arguments

`e1, x`	character vector (or an object coercible to) whose elements are to be duplicated
`e2, times`	numeric vector giving the number of times to repeat the corresponding strings

Details

Both arguments are recycled if necessary.

The `` `%x*%` `` operator mimics a vectorised version of Python's `` `*` `` for strings (`str.__mul__`).

Value

A character vector (in UTF-8).

`` `%x*%` `` and `strrep` preserve object attributes in a similar way as other Arithmetic operators.

Differences from Base R

Replacement for base `strrep` implemented with `stri_dup`.

- partial recycling with no warning "longer object length is not a multiple of shorter object length" **[fixed here]**
- base `strrep` seems to preserve only the `names` attribute, and only if the input is of type character (whilst `paste` preserves nothing) **[fixed]**
- overloading `` `*.character` `` has no effect in R, because S3 method dispatch is done internally with hard-coded support for character arguments. We could have replaced the generic `` `*` `` with the one that calls `UseMethod`, but it feels like a too intrusive solution **[fixed by introducing ‘`%x+%`‘ operator]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at `https://stringx.gagolewski.com/`

Related function(s): `paste`, `sprintf`

Examples

```
x <- structure(c(A="a", B=NA, C="c"), attrib1="value1")
x %x*% 3
x %x*% 1:3
"a" %x*% 1:3
stringx::strrep(x, 3)
base::strrep(x, 3)
y <- matrix(1:6, nrow=2, dimnames=list(c("A", "B"), NULL))
y %x*% 1:2
stringx::strrep(y, 1:2)
base::strrep(y, 1:2)
```

`strsplit` *Split Strings into Tokens*

Description

Splits each string into chunks delimited by occurrences of a given pattern.

Usage

```
strsplit(
  x,
  pattern = split,
  ...,
  ignore_case = ignore.case,
  fixed = FALSE,
  perl = FALSE,
  useBytes = FALSE,
  ignore.case = FALSE,
  split
)
```

Arguments

`x`	character vector whose elements are to be examined
`pattern`	character vector of nonempty search patterns
`...`	further arguments to `stri_split`, e.g., `omit_empty`, `locale`, `dotall`
`ignore_case`	single logical value; indicates whether matching should be case-insensitive
`fixed`	single logical value; `FALSE` for matching with regular expressions (see about_search_regex); `TRUE` for fixed pattern matching (about_search_fixed); `NA` for the Unicode collation algorithm (about_search_coll)
`perl`, `useBytes`	not used (with a warning if attempting to do so) [DEPRECATED]
`ignore.case`	alias to the `ignore_case` argument [DEPRECATED]
`split`	alias to the `pattern` argument [DEPRECATED]

Details

This function is fully vectorised with respect to both arguments.

For splitting text into 'characters' (grapheme clusters), words, or sentences, use `stri_split_boundaries` instead.

Value

Returns a list of character vectors representing the identified tokens.

Differences from Base R

Replacements for base `strsplit` implemented with `stri_split`.

- base R implementation is not portable as it is based on the system PCRE or TRE library (e.g., some Unicode classes may not be available or matching thereof can depend on the current `LC_CTYPE` category **[fixed here]**
- not suitable for natural language processing **[fixed here – use `fixed=NA`]**
- two different regular expression libraries are used (and historically, ERE was used in place of TRE) **[here, ICU Java-like regular expression engine is only available, hence the `perl` argument has no meaning]**

- there are inconsistencies between the argument order and naming in grepl, strsplit, and startsWith (amongst others); e.g., where the needle can precede the haystack, the use of the forward pipe operator, |>, is less convenient **[fixed here]**
- grepl also features the ignore.case argument **[added here]**
- if split is a zero-length vector, it is treated as "", which extracts individual code points (which is not the best idea for natural language processing tasks) **[empty search patterns are not supported here, zero-length vectors are propagated correctly]**
- last empty token is removed from the output, but first is not **[fixed here – see also the** omit_empty **argument]**
- missing values in split are not propagated correctly **[fixed here]**
- partial recycling without the usual warning, not fully vectorised w.r.t. the split argument **[fixed here]**
- only the names attribute of x is preserved **[fixed here]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): paste, nchar, grepl, gsub, substr

Examples

```
stringx::strsplit(c(x="a, b", y="c,d,  e"), ",\\s*")
x <- strcat(c(
    "abc", "123", ",!.", "\U0001F4A9",
    "\U0001F64D\U0001F3FC\U0000200D\U00002642\U0000FE0F",
    "\U000026F9\U0001F3FF\U0000200D\U00002640\U0000FE0F",
    "\U0001F3F4\U000E0067\U000E0062\U000E0073\U000E0063\U000E0074\U000E007F"
))
# be careful when splitting into individual code points:
base::strsplit(x, "")  # stringx does not support this
stringx::strsplit(x, "(?s)(?=.)", omit_empty=TRUE)  # look-ahead for any char with dot-all
stringi::stri_split_boundaries(x, type="character")  # grapheme clusters
```

---

strtrans *Transliteration and Other Text Transforms*

Description

These functions can be used to translate characters, including case mapping and folding, script to script conversion, and Unicode normalisation.

Usage

```
strtrans(x, transform)

chartr2(x, pattern, replacement)

chartr(old, new, x)

tolower(x, locale = NULL)

toupper(x, locale = NULL)

casefold(x, upper = NA)
```

Arguments

`x`	character vector (or an object coercible to)
`transform`	single string with ICU general transform specifier, see `stri_trans_list`
`pattern, old`	single string
`replacement, new`	single string, preferably of the same length as `old`
`locale`	`NULL` or `""` for the default locale (see `stri_locale_get`) or a single string with a locale identifier, see `stri_locale_list`
`upper`	single logical value; switches between case folding (the default, `NA`), lower-, and upper-case

Details

`tolower` and `toupper` perform case mapping. `chartr2` (and [DEPRECATED] `chartr`) translate individual code points. `casefold` commits case folding. The new function `strtrans` applies general **ICU** transforms, see `stri_trans_general`.

Value

These functions return a character vector (in UTF-8). They preserve most attributes of `x`. Note that their base R counterparts drop all the attributes if not fed with character vectors.

Differences from Base R

Unlike their base R counterparts, the new `tolower` and `toupper` are locale-sensitive; see `stri_trans_tolower`.

The base `casefold` simply dispatches to `tolower` or `toupper` 'for compatibility with S-PLUS' (which was only crucial long time ago). The version implemented here, by default, performs the true case folding, whose purpose is to make two pieces of text that differ only in case identical, see `stri_trans_casefold`.

`chartr2` and [DEPRECATED] `chartr` are wrappers for `stri_trans_char`. Contrary to the base `chartr`, they always generate a warning when `old` and `new` are of different lengths. `chartr2` has argument order and naming consistent with `gsub`.

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Examples

```
strtrans(strcat(letters_bf), "Any-NFKD; Any-Upper")
strtrans(strcat(letters_bb[1:6]), "Any-Hex/C")
strtrans(strcat(letters_greek), "Greek-Latin")

toupper(letters_greek)
tolower(LETTERS_GREEK)

base::toupper("gro\u00DF")
stringx::toupper("gro\u00DF")

casefold("gro\u00DF")

x <- as.matrix(c(a="\u00DFpam ba\U0001D554on spam", b=NA))
chartr("\u00DF\U0001D554aba", "SCXBA", x)

toupper('i', locale='en_US')
toupper('i', locale='tr_TR')
```

strtrim *Shorten Strings to Specified Width*

Description

Right-trims strings so that they do not exceed a given width (as determined by stri_width).

Usage

```
strtrim(x, width)
```

Arguments

x	character vector whose elements are to be trimmed
width	numeric vector giving the widths to which the corresponding strings are to be trimmed

Details

Both arguments are recycled if necessary.

Not to be confused with `trimws`.

Might be useful when displaying strings using a monospaced font.

Value

Returns a character vector (in UTF-8). Preserves object attributes in a similar way as Arithmetic operators.

Differences from Base R

Replacement for base `strtrim` implemented with (special case of) `stri_sprintf`.

- both arguments are not recycled in an usual manner **[fixed here]**
- missing values are not allowed in `width` **[fixed here]**
- some emojis, combining characters and modifiers (e.g., skin tones) are not recognised properly **[fixed here]**
- attributes are only propagated from the 1st argument **[fixed]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at `https://stringx.gagolewski.com/`

Related function(s): `sprintf`, `substr`, `nchar`

Examples

```
base::strtrim("aaaaa", 1:3)
stringx::strtrim("aaaaa", 1:3)
x <- c(
    "\U0001F4A9",
    "\U0001F64D\U0001F3FC\U0000200D\U00002642\U0000FE0F",
    "\U0001F64D\U0001F3FB\U0000200D\U00002642",
    "\U000026F9\U0001F3FF\U0000200D\U00002640\U0000FE0F",
    "\U0001F3F4\U000E0067\U000E0062\U000E0073\U000E0063\U000E0074\U000E007F"
)
print(x)
base::strtrim(x, 2)
stringx::strtrim(x, 2)
```

`strwrap` *Word-Wrap Text*

Description

Splits each string into words which are then arranged to form text lines of mo more than a given width.

Usage

```
strwrap(
  x,
  width = 0.9 * getOption("width"),
  indent = 0,
  exdent = 0,
  prefix = "",
  simplify = TRUE,
  initial = prefix,
  locale = NULL
)
```

Arguments

`x`	character vector whose elements are to be word-wrapped
`width`	single integer; maximal total width of the code points per line (as determined by `stri_width`)
`indent`	single integer; first line indentation size
`exdent`	single integer; consequent lines indentation size
`prefix`	single string; prefix for each line except the first
`simplify`	see Value
`initial`	single string; prefix for the first line
`locale`	`NULL` or `""` for the default locale (see `stri_locale_get`) or a single string with a locale identifier, see `stri_locale_list`

Details

Might be useful when displaying strings using a monospaced font.

Value

If `simplify` is `FALSE`, a list of `length(x)` numeric vectors is returned.

Otherwise, the function yields a character vector (in UTF-8). Note that the length of the output may be different than that of the input.

Due to this, no attributes are preserved.

Differences from Base R

Replacement for base strwrap implemented with stri_wrap.

- missing values not propagated **[fixed here]**
- some emojis, combining characters and modifiers (e.g., skin tones) are not recognised properly **[fixed here]**
- what is considered a word does not depend on locale **[fixed here - using ICU's word break iterators]**
- multiple whitespaces between words are not preserved except after a dot, question mark, or exclamation mark, which leads to two spaces inserted **[changed here – any sequence of whitespaces considered word boundaries is converted to a single space]**
- a greedy word wrap algorithm is used, which may lead to high raggedness **[fixed here – using the Knuth-Plass method]**

Author(s)

Marek Gagolewski

References

D.E. Knuth, M.F. Plass, Breaking paragraphs into lines, *Software: Practice and Experience* 11(11), 1981, pp. 1119–1184.

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): sprintf, trimws, nchar

Examples

```
strwrap(paste0(
    strrep("az ", 20),
    strrep("\u0105\u20AC ", 20),
    strrep("\U0001F643 ", 20),
    strrep("\U0001F926\U0000200D\U00002642\U0000FE0F ", 20)
), width=60)
```

sub2 *Replace Pattern Occurrences*

Description

sub2 replaces the first pattern occurrence in each string with a given replacement string. gsub2 replaces all (i.e., 'globally') pattern matches.

Usage

```
sub2(x, pattern, replacement, ..., ignore_case = FALSE, fixed = FALSE)

gsub2(x, pattern, replacement, ..., ignore_case = FALSE, fixed = FALSE)

sub(
  pattern,
  replacement,
  x,
  ...,
  ignore.case = FALSE,
  fixed = FALSE,
  perl = FALSE,
  useBytes = FALSE
)

gsub(
  pattern,
  replacement,
  x,
  ...,
  ignore.case = FALSE,
  fixed = FALSE,
  perl = FALSE,
  useBytes = FALSE
)
```

Arguments

x	character vector with strings whose chunks are to be modified
pattern	character vector of nonempty search patterns
replacement	character vector with the corresponding replacement strings; in sub2 and gsub2, back-references (whenever fixed=FALSE) are indicated by $0..$99 and $<name>, whereas the base-R compatible sub and gsub, only allow \1..\9
...	further arguments to stri_replace_first or stri_replace_all, e.g., locale, dotall

`ignore_case`, `ignore.case`
: single logical value; indicates whether matching should be case-insensitive

`fixed`
: single logical value; `FALSE` for matching with regular expressions (see about_search_regex); `TRUE` for fixed pattern matching (about_search_fixed); `NA` for the Unicode collation algorithm (about_search_coll)

`perl`, `useBytes`
: not used (with a warning if attempting to do so) [DEPRECATED]

Details

Not to be confused with `substr`.

These functions are fully vectorised with respect to `x`, `pattern`, and `replacement`.

`gsub2` uses `vectorise_all=TRUE` because of the attribute preservation rules, `stri_replace_all` should be called directly if different behaviour is needed.

The [DEPRECATED] `sub` and [DEPRECATED] `gsub` simply call `sub2` and `gsub2` which have a cleaned-up argument list. Additionally, if `fixed=FALSE`, the back-references in `replacement` strings are converted to these accepted by the **ICU** regex engine.

Value

Both functions return a character vector. They preserve the attributes of the longest inputs (unless they are dropped due to coercion).

Differences from Base R

Replacements for base `sub` and `gsub` implemented with `stri_replace_first` and `stri_replace_all`, respectively.

- there are inconsistencies between the argument order and naming in `grepl`, `strsplit`, and `startsWith` (amongst others); e.g., where the needle can precede the haystack, the use of the forward pipe operator, `|>`, is less convenient **[fixed here]**
- base R implementation is not portable as it is based on the system PCRE or TRE library (e.g., some Unicode classes may not be available or matching thereof can depend on the current `LC_CTYPE` category **[fixed here]**
- not suitable for natural language processing **[fixed here – use `fixed=NA`]**
- two different regular expression libraries are used (and historically, ERE was used in place of TRE) **[here, ICU Java-like regular expression engine is only available, hence the `perl` argument has no meaning]**
- not vectorised w.r.t. `pattern` and `replacement` **[fixed here]**
- only 9 (unnamed) back-references can be referred to in the replacement strings **[fixed in `sub2` and `gsub2`]**
- `perl=TRUE` supports `\U`, `\L`, and `\E` in the replacement strings **[not available here]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): paste, nchar, grepl2, gregexpr2, gregextr2 strsplit, gsubstr

trimws for removing whitespaces (amongst others) from the start or end of strings

Examples

```
"change \U0001f602 me \U0001f603" |> gsub2("\\p{L}+", "O_O")

x <- c("mario", "Mario", "M\u00E1rio", "M\u00C1RIO", "Mar\u00EDa", "Rosario", NA)
sub2(x, "mario", "M\u00E1rio", fixed=NA, strength=1L)
sub2(x, "mario", "Mario", fixed=NA, strength=2L)

x <- "abcdefghijklmnopqrstuvwxyz"
p <- "(.)(.)(.)(.)(.)(.)(.)(.)(.)(.)(.)(.)(.)"
base::sub(p, "\\1\\9", x)
base::gsub(p, "\\1\\9", x)
base::gsub(p, "\\1\\9", x, perl=TRUE)
base::gsub(p, "\\1\\13", x)
sub2(x, p, "$1$13")
gsub2(x, p, "$1$13")
```

substr — *Extract or Replace Substrings*

Description

substr and substrl extract contiguous parts of given character strings. The former operates based on start and end positions while the latter is fed with substring lengths.

Their replacement versions allow for substituting parts of strings with new content.

gsubstr and gsubstrl allow for extracting or replacing multiple chunks from each string.

Usage

```
substr(x, start = 1L, stop = -1L)

substrl(
  x,
  start = 1L,
  length = attr(start, "match.length"),
  ignore_negative_length = FALSE
)

substr(x, start = 1L, stop = -1L) <- value
```

```
substrl(x, start = 1L, length = attr(start, "match.length")) <- value

gsubstr(x, start = list(1L), stop = list(-1L))

gsubstrl(
  x,
  start = list(1L),
  length = lapply(start, attr, "match.length"),
  ignore_negative_length = TRUE
)

gsubstr(x, start = list(1L), stop = list(-1L)) <- value

gsubstrl(x, start = list(1L), length = lapply(start, attr, "match.length")) <- value

substring(text, first = 1L, last = -1L)

substring(text, first = 1L, last = -1L) <- value
```

Arguments

`x, text` character vector whose parts are to be extracted/replaced

`start, first` numeric vector (for `substr`) or list of numeric vectors (for `gsubstr`) giving the start indexes; e.g., 1 denotes the first code point; negative indexes count from the end of a string, i.e., -1 is the last character

`stop, last` numeric vector (for `substr`) or list of numeric vectors (for `gsubstr`) giving the end indexes (inclusive); note that if the start position is farther than the end position, this indicates an empty substring therein (see Examples)

`length` numeric vector (for `substr`) or list of numeric vectors (for `gsubstr`) giving the substring lengths; negative lengths result in a missing value or empty vector (see `ignore_negative_length`) or the corresponding substring being unchanged

`ignore_negative_length` single logical value; whether negative lengths should be ignored or yield missing values

`value` character vector (for `substr`) or list of character vectors (for `gsubstr`) defining the replacements strings

Details

Not to be confused with `sub`.

`substring` is a [DEPRECATED] synonym for `substr`.

Note that these functions can break some meaningful Unicode code point sequences, e.g., when inputs are not normalised. For extracting initial parts of strings based on character width, see `strtrim`.

Note that `gsubstr` (and related functions) expect `start`, `stop`, `length`, and `value` to be lists. Non-list arguments will be converted by calling `as.list`. This is different from the default policy applied by `stri_sub_all`, which calls `list`.

Note that substrl and gsubstrl are interoperable with regexpr2 and gregexpr2, respectively, and hence can be considered as substituted for the [DEPRECATED] regmatches (which is more specialised).

Value

substr and substrl return a character vector (in UTF-8). gsubstr and gsubstrl return a list of character vectors.

Their replacement versions modify x 'in-place' (see Examples).

The attributes are copied from the longest arguments (similar to binary operators).

Differences from Base R

Replacements for and enhancements of base substr and substring implemented with stri_sub and stri_sub_all,

- substring is "for compatibility with S", but this should no longer matter **[here,** substring **is equivalent to** substr**; in a future version, using the former may result in a warning]**
- substr is not vectorised with respect to all the arguments (and substring is not fully vectorised wrt value) **[fixed here]**
- not all attributes are taken from the longest of the inputs **[fixed here]**
- partial recycling with no warning **[fixed here]**
- the replacement must be of the same length as the chunk being substituted **[fixed here]**
- negative indexes are silently treated as 1 **[changed here: negative indexes count from the end of the string]**
- replacement of different length than the extracted substring never changes the length of the string **[changed here – output length is input length minus length of extracted plus length of replacement]**
- regexpr (amongst others) return start positions and lengths of matches, but base substr only uses start and end **[fixed by introducing** substrl**]**
- there is no function to extract or replace multiple chunks in each string (other than regmatches that works on outputs generated by gregexpr et al.) **[fixed by introducing** gsubstrl**]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): strtrim, nchar, startsWith, endsWith, gregexpr

Examples

```
x <- "spam, spam, bacon, and spam"
base::substr(x, c(1, 13), c(4, 17))
base::substring(x, c(1, 13), c(4, 17))
substr(x, c(1, 13), c(4, 17))
substrl(x, c(1, 13), c(4, 5))

# replacement function used as an ordinary one - return a copy of x:
base::`substr<-`(x, 1, 4, value="jam")
`substr<-`(x, 1, 4, value="jam")
base::`substr<-`(x, 1, 4, value="porridge")
`substr<-`(x, 1, 4, value="porridge")

# interoperability with gregexpr2:
p <- "[\\w&&[^a]][\\w&&[^n]][\\w&&[^d]]\\w+"  # regex: all words but 'and'
gsubstrl(x, gregexpr2(x, p))
`gsubstrl<-`(x, gregexpr2(x, p), value=list(c("a", "b", "c", "d")))

# replacement function modifying x in-place:
substr(x, 1, 4) <- "eggs"
substr(x, 1, 0) <- "porridge, "        # prepend (start<stop)
substr(x, nchar(x)+1) <- " every day"  # append (start<stop)
print(x)
```

---

trimws *Trim Leading or Trailing Whitespaces*

Description

Removes whitespaces (or other code points as specified by the `whitespace` argument) from left, right, or both sides of each string.

Usage

```
trimws(x, which = "both", whitespace = "\\p{Wspace}")
```

Arguments

`x`	character vector whose elements are to be trimmed
`which`	single string; either `"both"`, `"left"`, or `"right"`; side(s) from which the code points matching the `whitespace` pattern are to be removed
`whitespace`	single string; specifies the set of Unicode code points for removal, see 'Character Classes' in about_search_regex for more details

Details

Not to be confused with strtrim.

Value

Returns a character vector (in UTF-8).

Differences from Base R

Replacement for base trimws implemented with stri_replace_all_regex (and not stri_trim, which uses a slightly different syntax for pattern specifiers).

- the default whitespace argument does not reflect the 'contemporary' definition of whitespaces (e.g., does not include zero-width spaces) **[fixed here]**
- base R implementation is not portable as it is based on the system PCRE library (e.g., some Unicode classes may not be available or matching thereof can depend on the current LC_CTYPE category) **[fixed here]**
- no sanity checks are performed on whitespace **[fixed here]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): sub

Examples

```
base::trimws("NAAAAANA!!!NANAAAAA", whitespace=NA)  # stringx raises an error
x <- "   :)\v\u00a0 \n\r\t"
base::trimws(x)
stringx::trimws(x)
```

---

xtfrm2 *Sort Strings*

Description

The sort method for objects of class character (sort.character) uses the locale-sensitive Unicode collation algorithm to arrange strings in a vector with regards to a chosen lexicographic order.

xtfrm2 and [DEPRECATED] xtfrm generate an integer vector that sort in the same way as its input, and hence can be used in conjunction with order or rank.

Usage

```
xtfrm2(x, ...)

## Default S3 method:
xtfrm2(x, ...)

## S3 method for class 'character'
xtfrm2(
  x,
  ...,
  locale = NULL,
  strength = 3L,
  alternate_shifted = FALSE,
  french = FALSE,
  uppercase_first = NA,
  case_level = FALSE,
  normalisation = FALSE,
  numeric = FALSE
)

xtfrm(x)

## Default S3 method:
xtfrm(x)

## S3 method for class 'character'
xtfrm(x)

## S3 method for class 'character'
sort(
  x,
  ...,
  decreasing = FALSE,
  na.last = NA,
  locale = NULL,
  strength = 3L,
  alternate_shifted = FALSE,
  french = FALSE,
  uppercase_first = NA,
  case_level = FALSE,
  normalisation = FALSE,
  numeric = FALSE
)
```

Arguments

`x`	character vector whose elements are to be sorted
`...`	further arguments passed to other methods

`locale` `NULL` or `""` for the default locale (see `stri_locale_get`) or a single string with a locale identifier, see `stri_locale_list`

`strength` see `stri_opts_collator`

`alternate_shifted` see `stri_opts_collator`

`french` see `stri_opts_collator`

`uppercase_first` see `stri_opts_collator`

`case_level` see `stri_opts_collator`

`normalisation` see `stri_opts_collator`

`numeric` see `stri_opts_collator`

`decreasing` single logical value; if `FALSE`, the ordering is nondecreasing (weakly increasing)

`na.last` single logical value; if `TRUE`, then missing values are placed at the end; if `FALSE`, they are put at the beginning; if `NA`, then they are removed from the output whatsoever.

Details

What 'xtfrm' stands for the current author does not know, but would appreciate someone's enlightening him.

Value

`sort.character` returns a character vector, with only the `names` attribute preserved. Note that the output vector may be shorter than the input one.

`xtfrm2.character` and `xtfrm.character` return an integer vector; most attributes are preserved.

Differences from Base R

Replacements for the default S3 methods `sort` and `xtfrm` for character vectors implemented with `stri_sort` and `stri_rank`.

- Collation in different locales is difficult and non-portable across platforms **[fixed here – using services provided by ICU]**
- Overloading `xtfrm.character` has no effect in R, because S3 method dispatch is done internally with hard-coded support for character arguments. Thus, we needed to replace the generic `xtfrm` with the one that calls `UseMethod` **[fixed here]**
- `xtfrm` does not support customisation of the linear ordering relation it is based upon **[fixed by introducing `...` argument to the new generic, `xtfrm2`]**
- Neither `order`, `rank`, nor `sort.list` is a generic, therefore they should have to be rewritten from scratch to allow the inclusion of our patches; interestingly, `order` even calls `xtfrm`, but only for classed objects **[not fixed here – see Examples for a workaround]**
- `xtfrm` for objects of type `character` does not preserve the names attribute (but does so for `numeric`) **[fixed here]**

- sort seems to preserve only the names attribute which makes sense if na.last is NA, because the resulting vector might be shorter **[not fixed here as it would break compatibility with other sorting methods]**
- Note that sort by default removes missing values whatsoever, whereas order has na.last=TRUE **[not fixed here as it would break compatibility with other sorting methods]**

Author(s)

Marek Gagolewski

See Also

The official online manual of **stringx** at https://stringx.gagolewski.com/

Related function(s): strcoll

Examples

```
x <- c("a1", "a100", "a101", "a1000", "a10", "a10", "a11", "a99", "a10", "a1")
base::sort.default(x)   # lexicographic sort
sort(x, numeric=TRUE)   # calls stringx:::sort.character
xtfrm2(x, numeric=TRUE)  # calls stringx:::xtfrm2.character

rank(xtfrm2(x, numeric=TRUE), ties.method="average")  # ranks with averaged ties
order(xtfrm2(x, numeric=TRUE))    # ordering permutation
x[order(xtfrm2(x, numeric=TRUE))] # equivalent to sort()

# order a data frame w.r.t. decreasing ids and increasing vals
d <- data.frame(vals=round(runif(length(x)), 1), ids=x)
d[order(-xtfrm2(d[["ids"]], numeric=TRUE), d[["vals"]]), ]
```

Index