# Package 'stringfish'

April 13, 2022

**Title** Alt String Implementation

**Version** 0.15.7

**Date** 2022-4-12

**Maintainer** Travers Ching <traversc@gmail.com>

**Description** Provides an extendable, performant and multithreaded 'alt-string' implementation backed by 'C++' vectors and strings.

**License** GPL-3

**Biarch** true

**Encoding** UTF-8

**Depends** R (>= 3.0.2)

**SystemRequirements** C++11, GNU make

**LinkingTo** Rcpp (>= 0.12.18.3), RcppParallel (>= 5.1.4)

**Imports** Rcpp, RcppParallel

**Suggests** qs, knitr, rmarkdown, usethis, dplyr, stringr, rlang

**VignetteBuilder** knitr

**RoxygenNote** 7.1.2



**URL** https://github.com/traversc/stringfish

**BugReports** https://github.com/traversc/stringfish/issues

**NeedsCompilation** yes

**Author** Travers Ching [aut, cre, cph],
Phillip Hazel [ctb] (Bundled PCRE2 code),
Zoltan Herczeg [ctb, cph] (Bundled PCRE2 code),
University of Cambridge [cph] (Bundled PCRE2 code),
Tilera Corporation [cph] (Stack-less Just-In-Time compiler bundled with PCRE2),
Yann Collet [ctb, cph] (Yann Collet is the author of the bundled xxHash code)

**Repository** CRAN

**Date/Publication** 2022-04-13 07:00:02 UTC

# R topics documented:



---

`convert_to_sf` *convert_to_sf*

---

## Description

Converts a character vector to a stringfish vector

## Usage

```
convert_to_sf(x)

sf_convert(x)
```

### Arguments

| | |
|---|---|
| x | A character vector |

### Details

Converts a character vector to a stringfish vector. The opposite of ‘materialize‘.

### Value

The converted character vector

### Examples

```
if(getRversion() >= "3.5.0") {
x <- convert_to_sf(letters)
}
```

---

## get_string_type *get_string_type*

---

### Description

Returns the type of the character vector

### Usage

```
get_string_type(x)
```

### Arguments

| | |
|---|---|
| x | the vector |

### Details

A function that returns the type of character vector. Possible values are "normal vector", "stringfish vector", "stringfish vector (materialized)" or "other alt-rep vector"

### Value

The type of vector

### Examples

```
if(getRversion() >= "3.5.0") {
x <- sf_vector(10)
get_string_type(x) # returns "stringfish vector"
x <- character(10)
get_string_type(x) # returns "normal vector"
}
```

materialize *materialize*

## Description

Materializes an alt-rep object

## Usage

```
materialize(x)
```

## Arguments

| | |
|---|---|
| x | An alt-rep object |

## Details

Materializes any alt-rep object and then returns it. Note: the object is materialized regardless of whether the return value is assigned to a variable.

## Value

x

## Examples

```
if(getRversion() >= "3.5.0") {
x <- sf_vector(10)
sf_assign(x, 1, "hello world")
sf_assign(x, 2, "another string")
x <- materialize(x)
}
```

---

random_strings *random_strings*

## Description

A function that generates random strings

## Usage

```
random_strings(N, string_size = 50, charset = "abcdefghijklmnopqrstuvwxyz",
                       vector_mode = "stringfish")
```

### Arguments

| | |
|---|---|
| N | The number of strings to generate |
| string_size | The length of the strings |
| charset | The characters used to generate the random strings (default: abcdefghijklmnopqrstuvwxyz) |
| vector_mode | The type of character vector to generate (either stringfish or normal, default: stringfish) |

### Details

The function uses the PCRE2 library, which is also used internally by R. Note: the order of paramters is switched compared to the ‘gsub‘ base R function, with subject being first. See also: https://www.pcre.org/current/doc/html/pcre2api.html for more documentation on match syntax.

### Value

A character vector of the random strings

### See Also

gsub

### Examples

```
if(getRversion() >= "3.5.0") {
set.seed(1)
x <- random_strings(1e6, 80, "ACGT", vector_mode = "stringfish")
}
```

---

## sf_assign *sf_assign*

---

### Description

Assigns a new string to a stringfish vector or any other character vector

### Usage

```
sf_assign(x, i, e)
```

### Arguments

| | |
|---|---|
| x | the vector |
| i | the index to assign to |
| e | the new string to replace at i in x |

## Details

A function to assign a new element to an existing character vector. If the the vector is a stringfish vector, it does so without materialization.

## Value

No return value, the function assigns an element to an existing stringfish vector

## Examples

```
if(getRversion() >= "3.5.0") {
x <- sf_vector(10)
sf_assign(x, 1, "hello world")
sf_assign(x, 2, "another string")
}
```

---

`sf_collapse` *sf_collapse*

---

## Description

Pastes a series of strings together separated by the ‘collapse‘ parameter

## Usage

```
sf_collapse(x, collapse)
```

## Arguments

| | |
|---|---|
| `x` | A character vector |
| `collapse` | A single string |

## Details

This works the same way as ‘paste0(x, collapse=collapse)‘

## Value

A single string with all values in ‘x‘ pasted together, separated by ‘collapse‘.

## See Also

paste0, paste

## Examples

```
if(getRversion() >= "3.5.0") {
x <- c("hello", "\\xe4\\xb8\\x96\\xe7\\x95\\x8c")
Encoding(x) <- "UTF-8"
sf_collapse(x, " ") # "hello world" in Japanese
sf_collapse(letters, "") # returns the alphabet
}
```

---

sf_compare *sf_compare*

---

## Description

Returns a logical vector testing equality of strings from two string vectors

## Usage

```
sf_compare(x, y, nthreads = getOption("stringfish.nthreads", 1L))

sf_equals(x, y, nthreads = getOption("stringfish.nthreads", 1L))
```

## Arguments

| | |
|---|---|
| x | A character vector of length 1 or the same non-zero length as y |
| y | Another character vector of length 1 or the same non-zero length as y |
| nthreads | Number of threads to use |

## Details

Note: the function tests for both string and encoding equality

## Value

A logical vector

## Examples

```
if(getRversion() >= "3.5.0") {
sf_compare(letters, "a")
}
```

sf_concat *sf_concat*

## Description

Appends vectors together

## Usage

```
sf_concat(...)

sfc(...)
```

## Arguments

| | |
|---|---|
| ... | Any number of vectors, coerced to character vector if necessary |

## Value

A concatenated stringfish vector

## Examples

```
if(getRversion() >= "3.5.0") {
sf_concat(letters, 1:5)
}
```

sf_ends *sf_ends*

## Description

A function for detecting a pattern at the end of a string

## Usage

```
sf_ends(subject, pattern, ...)
```

## Arguments

| | |
|---|---|
| subject | A character vector |
| pattern | A string to look for at the start |
| ... | Parameters passed to sf_grepl |

### Value

A logical vector true if there is a match, false if no match, NA is the subject was NA

### See Also

endsWith, sf_starts

### Examples

```
if(getRversion() >= "3.5.0") {
x <- c("alpha", "beta", "gamma", "delta", "epsilon")
sf_ends(x, "a")
}
```

---

## sf_grepl *sf_grepl*

---

### Description

A function that matches patterns and returns a logical vector

### Usage

```
sf_grepl(subject, pattern, encode_mode = "auto", fixed = FALSE,
nthreads = getOption("stringfish.nthreads", 1L))
```

### Arguments

| | |
|---|---|
| subject | The subject character vector to search |
| pattern | The pattern to search for |
| encode_mode | "auto", "UTF-8" or "byte". Determines multi-byte (UTF-8) characters or single-byte characters are used. |
| fixed | determines whether the pattern parameter should be interpreted literally or as a regular expression |
| nthreads | Number of threads to use |

### Details

The function uses the PCRE2 library, which is also used internally by R. The encoding is based on the pattern string (or forced via the encode_mode parameter). Note: the order of paramters is switched compared to the ‘grepl‘ base R function, with subject being first. See also: https://www.pcre.org/current/doc/html/pc for more documentation on match syntax.

### Value

A logical vector with the same length as subject

## See Also

grepl

## Examples

```
if(getRversion() >= "3.5.0") {
x <- sf_vector(10)
sf_assign(x, 1, "hello world")
pattern <- "^hello"
sf_grepl(x, pattern)
}
```

---

sf_gsub *sf_gsub*

---

## Description

A function that performs pattern substitution

## Usage

```
sf_gsub(subject, pattern, replacement, encode_mode = "auto", fixed = FALSE,
nthreads = getOption("stringfish.nthreads", 1L))
```

## Arguments

| | |
|---|---|
| subject | The subject character vector to search |
| pattern | The pattern to search for |
| replacement | The replacement string |
| encode_mode | "auto", "UTF-8" or "byte". Determines multi-byte (UTF-8) characters or single-byte characters are used. |
| fixed | determines whether the pattern parameter should be interpreted literally or as a regular expression |
| nthreads | Number of threads to use |

## Details

The function uses the PCRE2 library, which is also used internally by R. However, syntax may be slightly different. E.g.: capture groups: "\1" in R, but "$1" in PCRE2 (as in Perl). The encoding of the output is determined by the pattern (or forced using encode_mode parameter) and encodings should be compatible. E.g: mixing ASCII and UTF-8 is okay, but not UTF-8 and latin1. Note: the order of paramters is switched compared to the ‘gsub‘ base R function, with subject being first. See also: https://www.pcre.org/current/doc/html/pcre2api.html for more documentation on match syntax.

## Value

A stringfish vector of the replacement string

## See Also

gsub

## Examples

```
if(getRversion() >= "3.5.0") {
x <- "hello world"
pattern <- "^hello (.+)"
replacement <- "goodbye $1"
sf_gsub(x, pattern, replacement)
}
```

---

`sf_iconv` *sf_iconv*

---

## Description

Converts encoding of one character vector to another

## Usage

```
sf_iconv(x, from, to, nthreads = getOption("stringfish.nthreads", 1L))
```

## Arguments

| | |
|---|---|
| `x` | An alt-rep object |
| `from` | the encoding to assume of ‘x‘ |
| `nthreads` | Number of threads to use |
| `to` | the new encoding |

## Details

This is an analogue to the base R function ‘iconv‘. It converts a string from one encoding (e.g. latin1 or UTF-8) to another

## Value

the converted character vector as a stringfish vector

## See Also

iconv

## Examples

```
if(getRversion() >= "3.5.0") {
x <- "fa\xE7ile"
Encoding(x) <- "latin1"
sf_iconv(x, "latin1", "UTF-8")
}
```

---

sf_match *sf_match*

---

## Description

Returns a vector of the positions of x in table

## Usage

```
sf_match(x, table, nthreads = getOption("stringfish.nthreads", 1L))
```

## Arguments

| | |
|---|---|
| x | A character vector to search for in table |
| table | A character vector to be matched against x |
| nthreads | Number of threads to use |

## Details

Note: similarly to the base R function, long "table" vectors are not supported. This is due to the maximum integer value that can be returned (‘.Machine$integer.max‘)

## Value

An integer vector of the indicies of each x element’s position in table

## See Also

match

## Examples

```
if(getRversion() >= "3.5.0") {
sf_match("c", letters)
}
```

sf_nchar *sf_nchar*

## Description

Counts the number of characters in a character vector

## Usage

```
sf_nchar(x, type = "chars", nthreads = getOption("stringfish.nthreads", 1L))
```

## Arguments

| | |
|---|---|
| x | A character vector |
| type | The type of counting to perform ("chars" or "bytes", default: "chars") |
| nthreads | Number of threads to use |

## Details

Returns the number of characters per string. The type of counting only matters for UTF-8 strings, where a character can be represented by multiple bytes.

## Value

An integer vector of the number of characters

## See Also

nchar

## Examples

```
if(getRversion() >= "3.5.0") {
x <- "fa\xE7ile"
Encoding(x) <- "latin1"
x <- sf_iconv(x, "latin1", "UTF-8")
}
```

sf_paste *sf_paste*

## Description

Pastes a series of strings together

## Usage

```
sf_paste(..., sep = "", nthreads = getOption("stringfish.nthreads", 1L))
```

## Arguments

| | |
|---|---|
| ... | Any number of character vector strings |
| sep | The seperating string between strings |
| nthreads | Number of threads to use |

## Details

This works the same way as ‘paste0(..., sep=sep)‘

## Value

A character vector where elements of the arguments are pasted together

## See Also

paste0, paste

## Examples

```
if(getRversion() >= "3.5.0") {
x <- letters
y <- LETTERS
sf_paste(x,y, sep = ":")
}
```

## sf_readLines *sf_readLines*

### Description

A function that reads a file line by line

### Usage

```
sf_readLines(file, encoding = "UTF-8")
```

### Arguments

| | |
|---|---|
| file | The file name |
| encoding | The encoding to use (Default: UTF-8) |

### Details

A function for reading in text data using ‘std::ifstream‘.

### Value

A stringfish vector of the lines in a file

### See Also

readLines

### Examples

```
if(getRversion() >= "3.5.0") {
file <- tempfile()
sf_writeLines(letters, file)
sf_readLines(file)
}
```

## sf_split *sf_split*

### Description

A function to split strings by a delimiter

### Usage

```
sf_split(subject, split, encode_mode = "auto", fixed = FALSE,
nthreads = getOption("stringfish.nthreads", 1L))
```

### Arguments

| | |
|---|---|
| subject | A character vector |
| split | A delimiter to split the string by |
| encode_mode | "auto", "UTF-8" or "byte". Determines multi-byte (UTF-8) characters or single-byte characters are used. |
| fixed | determines whether the split parameter should be interpreted literally or as a regular expression |
| nthreads | Number of threads to use |

### Value

A list of stringfish character vectors

### See Also

strsplit

### Examples

```
if(getRversion() >= "3.5.0") {
sf_split(datasets::state.name, "\\s") # split U.S. state names by any space character
}
```

---

## sf_starts *sf_starts*

---

### Description

A function for detecting a pattern at the start of a string

### Usage

```
sf_starts(subject, pattern, ...)
```

### Arguments

| | |
|---|---|
| subject | A character vector |
| pattern | A string to look for at the start |
| ... | Parameters passed to sf_grepl |

### Value

A logical vector true if there is a match, false if no match, NA is the subject was NA

### See Also

startsWith, sf_ends

## Examples

```
if(getRversion() >= "3.5.0") {
x <- c("alpha", "beta", "gamma", "delta", "epsilon")
sf_starts(x, "a")
}
```

---

sf_substr *sf_substr*

---

## Description

Extracts substrings from a character vector

## Usage

```
sf_substr(x, start, stop, nthreads = getOption("stringfish.nthreads", 1L))
```

## Arguments

| | |
|---|---|
| x | A character vector |
| start | The begining to extract from |
| stop | The end to extract from |
| nthreads | Number of threads to use |

## Details

This works the same way as 'substr', but in addition allows negative indexing. Negative indicies count backwards from the end of the string, with -1 being the last character.

## Value

A stringfish vector of substrings

## See Also

substr

## Examples

```
if(getRversion() >= "3.5.0") {
x <- c("fa\xE7ile", "hello world")
Encoding(x) <- "latin1"
x <- sf_iconv(x, "latin1", "UTF-8")
sf_substr(x, 4, -1) # extracts from the 4th character to the last
## [1] "ile"  "lo world"
}
```

## sf_tolower *sf_tolower*

### Description

A function converting a string to all lowercase

### Usage

```
sf_tolower(x)
```

### Arguments

| | |
|---|---|
| x | A character vector |

### Details

Note: the function only converts ASCII characters.

### Value

A stringfish vector where all uppercase is converted to lowercase

### See Also

tolower

### Examples

```
if(getRversion() >= "3.5.0") {
x <- LETTERS
sf_tolower(x)
}
```

## sf_toupper *sf_toupper*

### Description

A function converting a string to all uppercase

### Usage

```
sf_toupper(x)
```

### Arguments

| | |
|---|---|
| x | A character vector |

### Details

Note: the function only converts ASCII characters.

### Value

A stringfish vector where all lowercase is converted to uppercase

### See Also

toupper

### Examples

```
if(getRversion() >= "3.5.0") {
x <- letters
sf_toupper(x)
}
```

---

## sf_trim *sf_trim*

---

### Description

A function to remove leading/trailing whitespace

### Usage

```
sf_trim(subject, which = c("both", "left", "right"), whitespace = "[ \\t\\r\\n]", ...)
```

### Arguments

| | |
|---|---|
| subject | A character vector |
| which | "both", "left", or "right" determines which white space is removed |
| whitespace | Whitespace characters (default: "[ \\t\\r\\n]") |
| ... | Parameters passed to sf_gsub |

### Value

A stringfish vector of trimmed whitespace

### See Also

trimws

## Examples

```
if(getRversion() >= "3.5.0") {
x <- c(" alpha ", " beta", " gamma ", "delta ", "epsilon ")
sf_trim(x)
}
```

---

`sf_vector` *sf_vector*

---

## Description

Creates a new stringfish vector

## Usage

```
sf_vector(len)
```

## Arguments

| | |
|---|---|
| `len` | length of the new vector |

## Details

This function creates a new stringfish vector, an alt-rep character vector backed by a C++ "std::vector" as the internal memory representation. The vector type is "sfstring", which is a simple C++ class containing a "std::string" and a single byte (uint8_t) representing the encoding.

## Value

A new (empty) stringfish vector

## Examples

```
if(getRversion() >= "3.5.0") {
x <- sf_vector(10)
sf_assign(x, 1, "hello world")
sf_assign(x, 2, "another string")
}
```

## sf_writeLines *sf_writeLines*

### Description

A function that reads a file line by line

### Usage

```
sf_writeLines(text, file, sep = "\n", na_value = "NA", encode_mode = "UTF-8")
```

### Arguments

| | |
|---|---|
| text | A character to write to file |
| file | Name of the file to write to |
| sep | The line separator character(s) |
| na_value | What to write in case of a NA string |
| encode_mode | "UTF-8" or "byte". If "UTF-8", all strings are re-encoded as UTF-8. |

### Details

A function for writing text data using ‘std::ofstream‘.

### See Also

writeLines

### Examples

```
if(getRversion() >= "3.5.0") {
file <- tempfile()
sf_writeLines(letters, file)
sf_readLines(file)
}
```

## string_identical *string_identical*

### Description

A stricter comparison of string equality

### Usage

```
string_identical(x, y)
```

## Arguments

| | |
|---|---|
| x | A character vector |
| y | Another character to compare to x |

## Value

TRUE if strings are identical, including encoding

## See Also

identical

# Index