Package 'snakecase'

May 26, 2019

Version 0.11.0

Date 2019-05-25

Title Convert Strings into any Case

Description A consistent, flexible and easy to use tool to parse and convert strings into cases like snake or camel among others.

Maintainer Malte Grosser <malte.grosser@gmail.com>

Depends R (>= 3.2)

Imports stringr, stringi

Suggests testthat, covr, tibble, purrrlyr, knitr, rmarkdown, magrittr

URL https://github.com/Tazinho/snakecase

BugReports https://github.com/Tazinho/snakecase/issues

Encoding UTF-8

License GPL-3

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Malte Grosser [aut, cre]

Repository CRAN

Date/Publication 2019-05-25 22:50:03 UTC

R topics documented:

`abbreviation_internal` *Internal abbreviation marker, marks abbreviations with an underscore behind. Useful if* `parsing_option` *1 is needed, but some abbreviations need* `parsing_option` *2.*

Description

Internal abbreviation marker, marks abbreviations with an underscore behind. Useful if `parsing_option` 1 is needed, but some abbreviations need `parsing_option` 2.

Usage

```
abbreviation_internal(string, abbreviations = NULL)
```

Arguments

`string`	A string (for example names of a data frame).
`abbreviations`	character with (uppercase) abbreviations. This marks abbreviations with an underscore behind (in front of the parsing). Useful if `parsing_option` 1 is needed, but some abbreviations need `parsing_option` 2.

Value

A character vector.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

`caseconverter` *Specific case converter shortcuts*

Description

Wrappers around `to_any_case()`

Usage

```
to_snake_case(string, abbreviations = NULL, sep_in = "[^[:alnum:]]",
  parsing_option = 1, transliterations = NULL, numerals = "middle",
  sep_out = NULL, unique_sep = NULL, empty_fill = NULL,
  prefix = "", postfix = "")

to_lower_camel_case(string, abbreviations = NULL,
  sep_in = "[^[:alnum:]]", parsing_option = 1,
  transliterations = NULL, numerals = "middle", sep_out = NULL,
  unique_sep = NULL, empty_fill = NULL, prefix = "", postfix = "")

to_upper_camel_case(string, abbreviations = NULL,
  sep_in = "[^[:alnum:]]", parsing_option = 1,
  transliterations = NULL, numerals = "middle", sep_out = NULL,
  unique_sep = NULL, empty_fill = NULL, prefix = "", postfix = "")

to_screaming_snake_case(string, abbreviations = NULL,
  sep_in = "[^[:alnum:]]", parsing_option = 1,
  transliterations = NULL, numerals = "middle", sep_out = NULL,
  unique_sep = NULL, empty_fill = NULL, prefix = "", postfix = "")

to_parsed_case(string, abbreviations = NULL, sep_in = "[^[:alnum:]]",
  parsing_option = 1, transliterations = NULL, numerals = "middle",
  sep_out = NULL, unique_sep = NULL, empty_fill = NULL,
  prefix = "", postfix = "")

to_mixed_case(string, abbreviations = NULL, sep_in = "[^[:alnum:]]",
  parsing_option = 1, transliterations = NULL, numerals = "middle",
  sep_out = NULL, unique_sep = NULL, empty_fill = NULL,
  prefix = "", postfix = "")

to_lower_upper_case(string, abbreviations = NULL,
  sep_in = "[^[:alnum:]]", parsing_option = 1,
  transliterations = NULL, numerals = "middle", sep_out = NULL,
  unique_sep = NULL, empty_fill = NULL, prefix = "", postfix = "")

to_upper_lower_case(string, abbreviations = NULL,
  sep_in = "[^[:alnum:]]", parsing_option = 1,
  transliterations = NULL, numerals = "middle", sep_out = NULL,
  unique_sep = NULL, empty_fill = NULL, prefix = "", postfix = "")

to_swap_case(string, abbreviations = NULL, sep_in = "[^[:alnum:]]",
  parsing_option = 1, transliterations = NULL, numerals = "middle",
  sep_out = NULL, unique_sep = NULL, empty_fill = NULL,
  prefix = "", postfix = "")

to_sentence_case(string, abbreviations = NULL, sep_in = "[^[:alnum:]]",
  parsing_option = 1, transliterations = NULL, numerals = "middle",
```

```
  sep_out = NULL, unique_sep = NULL, empty_fill = NULL,
  prefix = "", postfix = "")

to_random_case(string, abbreviations = NULL, sep_in = "[^[:alnum:]]",
  parsing_option = 1, transliterations = NULL, numerals = "middle",
  sep_out = NULL, unique_sep = NULL, empty_fill = NULL,
  prefix = "", postfix = "")

to_title_case(string, abbreviations = NULL, sep_in = "[^[:alnum:]]",
  parsing_option = 1, transliterations = NULL, numerals = "middle",
  sep_out = NULL, unique_sep = NULL, empty_fill = NULL,
  prefix = "", postfix = "")
```

Arguments

`string`	A string (for example names of a data frame).
`abbreviations`	character. (Case insensitive) matched abbreviations are surrounded by underscores. In this way, they can get recognized by the parser. This is useful when e.g. `parsing_option` 1 is needed for the use case, but some abbreviations but some substrings would require `parsing_option` 2. Furthermore, this argument also specifies the formatting of abbreviations in the output for the cases title, mixed, lower and upper camel. E.g. for upper camel the first letter is always in upper case, but when the abbreviation is supplied in upper case, this will also be visible in the output. Use this feature with care: One letter abbreviations and abbreviations next to each other are hard to read and also not easy to parse for further processing.
`sep_in`	(short for separator input) if character, is interpreted as a regular expression (wrapped internally into `stringr::regex()`). The default value is a regular expression that matches any sequence of non-alphanumeric values. All matches will be replaced by underscores (additionally to `"_"` and `" "`, for which this is always true, even if `NULL` is supplied). These underscores are used internally to split the strings into substrings and specify the word boundaries.
`parsing_option`	An integer that will determine the parsing_option. • 1: `"RRRStudio" -> "RRR_Studio"` • 2: `"RRRStudio" -> "RRRS_tudio"` • 3: `"RRRStudio" -> "RRRSStudio"`. This will become for example `"Rrrstudio"` when we convert to lower camel case. • -1, -2, -3: These `parsing_options`'s will suppress the conversion after non-alphanumeric values. • 0: no parsing
`transliterations`	A character vector (if not `NULL`). The entries of this argument need to be elements of `stringi::stri_trans_list()` (like "Latin-ASCII", which is often useful) or names of lookup tables (currently only "german" is supported). In the order of the entries the letters of the input string will be transliterated via `stringi::stri_trans_general()` or replaced via the matches of the lookup table. When named character elements are supplied as part of 'transliterations',

anything that matches the names is replaced by the corresponding value. You should use this feature with care in case of `case = "parsed"`, `case = "internal_parsing"` and `case = "none"`, since for upper case letters, which have transliterations/replacements of length 2, the second letter will be transliterated to lowercase, for example Oe, Ae, Ss, which might not always be what is intended. In this case you can make usage of the option to supply named elements and specify the transliterations yourself.

`numerals` A character specifying the alignment of numerals (`"middle"`, `left`, `right` or `asis`). I.e. `numerals = "left"` ensures that no output separator is in front of a digit.

`sep_out` (short for separator output) String that will be used as separator. The defaults are `"_"` and `""`, regarding the specified `case`. When `length(sep_out) > 1`, the last element of `sep_out` gets recycled and separators are incorporated per string according to their order.

`unique_sep` A string. If not `NULL`, then duplicated names will get a suffix integer in the order of their appearance. The suffix is separated by the supplied string to this argument.

`empty_fill` A string. If it is supplied, then each entry that matches "" will be replaced by the supplied string to this argument.

`prefix` prefix (string).

`postfix` postfix (string).

Value

A character vector according the specified parameters above.

A character vector according the specified target case.

Note

caseconverters are vectorised over `string`, `sep_in`, `sep_out`, `empty_fill`, `prefix` and `postfix`.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

Malte Grosser, <malte.grosser@gmail.com>

See Also

snakecase on github, `to_any_case` for flexible high level conversion and more examples.

Examples

```
strings <- c("this Is a Strange_string", "AND THIS ANOTHER_One", NA)

to_snake_case(strings)
to_lower_camel_case(strings)
to_upper_camel_case(strings)
to_screaming_snake_case(strings)
```

```
to_lower_upper_case(strings)
to_upper_lower_case(strings)
to_parsed_case(strings)
to_mixed_case(strings)
to_swap_case(strings)
to_sentence_case(strings)
to_random_case(strings)
to_title_case(strings)
```

`check_design_rule` *Internal helper to test the design rules for any string and setting of* `to_any_case()`

Description

Internal helper to test the design rules for any string and setting of `to_any_case()`

Usage

```
check_design_rule(string, sep_in = NULL, transliterations = NULL,
  sep_out = NULL, prefix = "", postfix = "", unique_sep = NULL,
  empty_fill = NULL, parsing_option = 1)
```

Arguments

`string`	A string (for example names of a data frame).
`sep_in`	String that will be wrapped internally into `stringr::regex()`. All matches will be treated as additional splitting parameters besides the default ones (`"_"` and `" "`), when parsing the input string.
`transliterations`	A character vector (if not `NULL`). The entries of this argument need to be elements of `stringi::stri_trans_list()` (like "Latin-ASCII", which is often useful) or names of lookup tables (currently only "german" is supported). In the order of the entries the letters of the input string will be transliterated via `stringi::stri_trans_general()` or replaced via the matches of the lookup table.
`sep_out`	String that will be used as separator. The defaults are `"_"` and `""`, regarding the specified `case`.
`prefix`	prefix (string).
`postfix`	postfix (string).
`unique_sep`	A string. If it is supplied, then duplicated names will get a suffix integer in the order of their appearance. The suffix is separated by the supplied string to this argument.

`empty_fill` A string. If it is supplied, then each entry that matches "" will be replaced by the supplied string to this argument.

`parsing_option` An integer that will determine the parsing_option.

- 1: `RRRStudio -> RRR_Studio`
- 2: `RRRStudio -> RRRS_tudio`
- 3: parses at the beginning like option 1 and the rest like option 2.
- 4: parses at the beginning like option 2 and the rest like option 1.
- 5: parses like option 1 but suppresses "_" around non special characters. In this way case conversion won't apply after these characters. See examples.
- 6: parses like option 1, but digits directly behind/in front non-digits, will stay as is.
- any other integer <= 0: no parsing"

Value

A character vector separated by underscores, containing the parsed string.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

`parsing_helpers` *Parsing helpers*

Description

Mainly for usage within `to_parsed_case_internal`

Usage

```
parse1_pat_cap_smalls(string)

parse2_pat_digits(string)

parse3_pat_caps(string)

parse4_pat_cap(string)

parse5_pat_non_alnums(string)

parse6_mark_digits(string)

parse7_pat_caps_smalls(string)

parse8_pat_smalls_after_non_alnums(string)
```

Arguments

string	A string.

Value

A partly parsed character vector.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

---

preprocess_internal *Internal function that replaces regex matches with underscores*

Description

Internal function that replaces regex matches with underscores

Usage

```
preprocess_internal(string, sep_in)
```

Arguments

string	A string.
sep_in	(short for separator input) A regex supplied as a character (if not NULL), which will be wrapped internally into stringr::regex(). All matches will be replaced by underscores (additionally to "_" and " ", for which this is always true). Underscores can later turned into another separator via postprocess.

Value

A character containing the parsed string.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

relevant *Internal helper for "lower_upper", "upper_lower". This helper returns a logical vector with TRUE for the first and every second string of those which contain an alphabetic character*

Description

Internal helper for "lower_upper", "upper_lower". This helper returns a logical vector with TRUE for the first and every second string of those which contain an alphabetic character

Usage

```
relevant(string)
```

Arguments

string	A string (for example names of a data frame).

Value

A logical vector.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

replace_special_characters_internal *Internal helper to replace special characters.*

Description

Internal helper to replace special characters.

Usage

```
replace_special_characters_internal(string, transliterations, case)
```

Arguments

string	A string (for example names of a data frame).
transliterations	A character vector (if not NULL). The entries of this argument need to be elements of stringi::stri_trans_list() (like "Latin-ASCII", which is often useful) or names of lookup tables (currently only "german" is supported). In the order of the entries the letters of the input string will be transliterated via stringi::stri_trans_general() or replaced via the matches of the lookup table. When named character elements are supplied as part of 'transliterations', anything that matches the names is replaced by the corresponding value. You should use this feature with care in case of case = "parsed", case = "internal_parsing" and case = "none", since for upper case letters, which have transliterations/replacements of length 2, the second letter will be transliterated to lowercase, for example Oe, Ae, Ss, which might not always be what is intended. In this case you can make usage of the option to supply named elements and specify the transliterations yourself.
case	Length one character, from the input options of to_any_case.

Value

A character vector.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

---

to_any_case *General case conversion*

Description

Function to convert strings to any case

Usage

```
to_any_case(string, case = c("snake", "small_camel", "big_camel",
  "screaming_snake", "parsed", "mixed", "lower_upper", "upper_lower",
  "swap", "all_caps", "lower_camel", "upper_camel", "internal_parsing",
  "none", "flip", "sentence", "random", "title"), abbreviations = NULL,
  sep_in = "[^[:alnum:]]", parsing_option = 1,
  transliterations = NULL, numerals = c("middle", "left", "right",
  "asis", "tight"), sep_out = NULL, unique_sep = NULL,
  empty_fill = NULL, prefix = "", postfix = "")
```

Arguments

`string` A string (for example names of a data frame).

`case` The desired target case, provided as one of the following:

- snake_case: `"snake"`
- lowerCamel: `"lower_camel"` or `"small_camel"`
- UpperCamel: `"upper_camel"` or `"big_camel"`
- ALL_CAPS: `"all_caps"` or `"screaming_snake"`
- lowerUPPER: `"lower_upper"`
- UPPERlower: `"upper_lower"`
- Sentence case: `"sentence"`
- Title Case: `"title"` - This one is basically the same as sentence case, but in addition it is wrapped into `tools::toTitleCase` and any `abbreviations` are always turned into upper case.

There are five "special" cases available:

- `"parsed"`: This case is underlying all other cases. Every substring a string consists of becomes surrounded by an underscore (depending on the `parsing_option`). Underscores at the start and end are trimmed. No lower or upper case pattern from the input string are changed.
- `"mixed"`: Almost the same as `case = "parsed"`. Every letter which is not at the start or behind an underscore is turned into lowercase. If a substring is set as an abbreviation, it will be turned into upper case.
- `"swap"`: Upper case letters will be turned into lower case and vice versa. Also `case = "flip"` will work. Doesn't work with any of the other arguments except `unique_sep`, `empty_fill`, `prefix` and `postfix`.
- `"random"`: Each letter will be randomly turned into lower or upper case. Doesn't work with any of the other arguments except `unique_sep`, `empty_fill`, `prefix` and `postfix`.
- `"none"`: Neither parsing nor case conversion occur. This case might be helpful, when one wants to call the function for the quick usage of the other parameters. To suppress replacement of spaces to underscores set `sep_in = NULL`. Works with `sep_in`, `transliterations`, `sep_out`, `prefix`, `postfix`, `empty_fill` and `unique_sep`.
- `"internal_parsing"`: This case is returning the internal parsing (suppressing the internal protection mechanism), which means that alphanumeric characters will be surrounded by underscores. It should only be used in very rare use cases and is mainly implemented to showcase the internal workings of `to_any_case()`

`abbreviations` character. (Case insensitive) matched abbreviations are surrounded by underscores. In this way, they can get recognized by the parser. This is useful when e.g. `parsing_option` 1 is needed for the use case, but some abbreviations but some substrings would require `parsing_option` 2. Furthermore, this argument also specifies the formatting of abbreviations in the output for the cases title, mixed, lower and upper camel. E.g. for upper camel the first letter is always in upper case, but when the abbreviation is supplied in upper case, this will also be visible in the output.

Use this feature with care: One letter abbreviations and abbreviations next to each other are hard to read and also not easy to parse for further processing.

`sep_in` (short for separator input) if character, is interpreted as a regular expression (wrapped internally into `stringr::regex()`). The default value is a regular expression that matches any sequence of non-alphanumeric values. All matches will be replaced by underscores (additionally to `"_"` and `" "`, for which this is always true, even if `NULL` is supplied). These underscores are used internally to split the strings into substrings and specify the word boundaries.

`parsing_option` An integer that will determine the parsing_option.

- 1: `"RRRStudio" -> "RRR_Studio"`
- 2: `"RRRStudio" -> "RRRS_tudio"`
- 3: `"RRRStudio" -> "RRRSStudio"`. This will become for example `"Rrrstudio"` when we convert to lower camel case.
- -1, -2, -3: These `parsing_options`'s will suppress the conversion after non-alphanumeric values.
- 0: no parsing

`transliterations`

A character vector (if not `NULL`). The entries of this argument need to be elements of `stringi::stri_trans_list()` (like "Latin-ASCII", which is often useful) or names of lookup tables (currently only "german" is supported). In the order of the entries the letters of the input string will be transliterated via `stringi::stri_trans_general()` or replaced via the matches of the lookup table. When named character elements are supplied as part of 'transliterations', anything that matches the names is replaced by the corresponding value. You should use this feature with care in case of `case = "parsed"`, `case = "internal_parsing"` and `case = "none"`, since for upper case letters, which have transliterations/replacements of length 2, the second letter will be transliterated to lowercase, for example Oe, Ae, Ss, which might not always be what is intended. In this case you can make usage of the option to supply named elements and specify the transliterations yourself.

`numerals` A character specifying the alignment of numerals (`"middle"`, `left`, `right`, `asis` or `tight`). I.e. `numerals = "left"` ensures that no output separator is in front of a digit.

`sep_out` (short for separator output) String that will be used as separator. The defaults are `"_"` and `""`, regarding the specified `case`. When `length(sep_out) > 1`, the last element of `sep_out` gets recycled and separators are incorporated per string according to their order.

`unique_sep` A string. If not `NULL`, then duplicated names will get a suffix integer in the order of their appearance. The suffix is separated by the supplied string to this argument.

`empty_fill` A string. If it is supplied, then each entry that matches "" will be replaced by the supplied string to this argument.

`prefix` prefix (string).

`postfix` postfix (string).

Value

A character vector according the specified parameters above.

Note

`to_any_case()` is vectorised over `string`, `sep_in`, `sep_out`, `empty_fill`, `prefix` and `postfix`.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

See Also

snakecase on github or `caseconverter` for some handy shortcuts.

Examples

```
### abbreviations
to_snake_case(c("HHcity", "newUSElections"), abbreviations = c("HH", "US"))
to_upper_camel_case("succesfullGMBH", abbreviations = "GmbH")
to_title_case("succesfullGMBH", abbreviations = "GmbH")

### sep_in (input separator)
string <- "R.St\u00FCdio: v.1.0.143"
to_any_case(string)
to_any_case(string, sep_in = ":|\\.")
to_any_case(string, sep_in = ":|(?<!\\d)\\.")

### parsing_option
# the default option makes no sense in this setting
to_parsed_case("HAMBURGcity", parsing_option = 1)
# so the second parsing option is the way to address this example
to_parsed_case("HAMBURGcity", parsing_option = 2)
# By default (option 1) characters are converted after non alpha numeric characters.
# To suppress this behaviour add a minus to the parsing_option
to_upper_camel_case("lookBehindThe.dot", parsing_option = -1)
# For some exotic cases parsing option 3 might be of interest
to_parsed_case("PARSingOption3", parsing_option = 3)
# There may be reasons to suppress the parsing
to_any_case("HAMBURGcity", parsing_option = 0)

### transliterations
to_any_case("\u00E4ngstlicher Has\u00EA", transliterations = c("german", "Latin-ASCII"))

### case
strings <- c("this Is a Strange_string", "AND THIS ANOTHER_One")
to_any_case(strings, case = "snake")
to_any_case(strings, case = "lower_camel") # same as "small_camel"
to_any_case(strings, case = "upper_camel") # same as "big_camel"
to_any_case(strings, case = "all_caps") # same as "screaming_snake"
to_any_case(strings, case = "lower_upper")
to_any_case(strings, case = "upper_lower")
```

```
to_any_case(strings, case = "sentence")
to_any_case(strings, case = "title")
to_any_case(strings, case = "parsed")
to_any_case(strings, case = "mixed")
to_any_case(strings, case = "swap")
to_any_case(strings, case = "random")
to_any_case(strings, case = "none")
to_any_case(strings, case = "internal_parsing")

### numerals
to_snake_case("species42value 23month 7-8", numerals = "asis")
to_snake_case("species42value 23month 7-8", numerals = "left")
to_snake_case("species42value 23month 7-8", numerals = "right")
to_snake_case("species42value 23month 7-8", numerals = "middle")
to_snake_case("species42value 23month 7-8", numerals = "tight")

### sep_out (output separator)
string <- c("lowerCamelCase", "ALL_CAPS", "I-DontKNOWWhat_thisCASE_is")
to_snake_case(string, sep_out = ".")
to_mixed_case(string, sep_out = " ")
to_screaming_snake_case(string, sep_out = "=")

### empty_fill
to_any_case(c("","",""), empty_fill = c("empty", "empty", "also empty"))

### unique_sep
to_any_case(c("same", "same", "same", "other"), unique_sep = c(">"))

### prefix and postfix
to_upper_camel_case("some_path", sep_out = "//",
  prefix = "USER://", postfix = ".exe")
```

`to_parsed_case_internal` *Internal parser, which is relevant for preprocessing, parsing and parsing options*

Description

Internal parser, which is relevant for preprocessing, parsing and parsing options

Usage

```
to_parsed_case_internal(string, parsing_option = 1L, numerals,
  abbreviations, sep_in)
```

Arguments

`string`	A string.
`parsing_option`	An integer that will determine the parsing option. • 1: `RRRStudio -> RRR_Studio` • 2: `RRRStudio -> RRRS_tudio` • 3: parses like option 1 but suppresses "_" around non alpha-numeric characters. In this way this option suppresses splits and resulting case conversion after these characters. • any other integer <= 0: no parsing"
`numerals`	A character specifying the alignment of numerals (`"middle"`, `left`, `right` or `asis`). I.e. `numerals = "left"` ensures that no output separator is in front of a digit.
`abbreviations`	A character string specifying abbreviations that should be marked to be recognized by later parsing.
`sep_in`	A character (regular expression) used to specify input separators.

Value

A character vector separated by underscores, containing the parsed string.

Author(s)

Malte Grosser, <malte.grosser@gmail.com>

Index