

Package ‘assertive.strings’

October 12, 2022

Type Package

Title Assertions to Check Properties of Strings

Version 0.0-3

Date 2016-03-08

Author Richard Cotton [aut, cre],
Aditya Bhagwat [ctb]

Maintainer Richard Cotton <richierocks@gmail.com>

Description A set of predicates and assertions for checking the properties of strings. This is mainly for use by other package developers who want to include run-time testing features in their own packages. End-users will usually want to use assertive directly.

URL <https://bitbucket.org/richierocks/assertive.strings>

BugReports <https://bitbucket.org/richierocks/assertive.strings/issues>

Depends R (>= 3.0.0)

Imports assertive.base (>= 0.0-2), assertive.types, stringi

Suggests testthat

License GPL (>= 3)

LazyLoad yes

LazyData yes

Acknowledgments Development of this package was partially funded by the Proteomics Core at Weill Cornell Medical College in Qatar <<http://qatar-weill.cornell.edu>>. The Core is supported by 'Biomedical Research Program' funds, a program funded by Qatar Foundation.

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-05-10 10:14:32

R topics documented:

assert_all_are_empty_character	2
assert_all_are_matching_fixed	4
assert_all_are_numeric_strings	6
assert_all_are_single_characters	7
character_to_list_of_integer_vectors	8
create_regex	9
d	10
matches_regex	10
recycle	11
strip_invalid_chars	12

Index	13
--------------	-----------

assert_all_are_empty_character

Does the input contain empty or missing strings?

Description

Checks for empty or missing strings.

Usage

```
assert_all_are_empty_character(x, severity = getOption("assertive.severity",
"stop"))
```

```
assert_any_are_empty_character(x, severity = getOption("assertive.severity",
"stop"))
```

```
assert_all_are_non_empty_character(x,
severity = getOption("assertive.severity", "stop"))
```

```
assert_any_are_non_empty_character(x,
severity = getOption("assertive.severity", "stop"))
```

```
assert_all_are_missing_or_empty_character(x,
severity = getOption("assertive.severity", "stop"))
```

```
assert_any_are_missing_or_empty_character(x,
severity = getOption("assertive.severity", "stop"))
```

```
assert_all_are_non_missing_nor_empty_character(x,
severity = getOption("assertive.severity", "stop"))
```

```
assert_any_are_non_missing_nor_empty_character(x,
severity = getOption("assertive.severity", "stop"))
```

```
assert_all_strings_are_not_missing_nor_empty(x,  
  severity = getOption("assertive.severity", "stop"))  
  
assert_any_strings_are_not_missing_nor_empty(x,  
  severity = getOption("assertive.severity", "stop"))  
  
assert_is_an_empty_string(x, severity = getOption("assertive.severity",  
  "stop"))  
  
assert_is_a_non_empty_string(x, severity = getOption("assertive.severity",  
  "stop"))  
  
assert_is_a_missing_or_empty_string(x,  
  severity = getOption("assertive.severity", "stop"))  
  
assert_is_a_non_missing_nor_empty_string(x,  
  severity = getOption("assertive.severity", "stop"))  
  
is_empty_character(x, .xname = get_name_in_parent(x))  
  
is_non_empty_character(x, .xname = get_name_in_parent(x))  
  
is_missing_or_empty_character(x, .xname = get_name_in_parent(x))  
  
is_non_missing_nor_empty_character(x, .xname = get_name_in_parent(x))  
  
is_not_missing_nor_empty_character(x)  
  
is_an_empty_string(x, .xname = get_name_in_parent(x))  
  
is_a_non_empty_string(x, .xname = get_name_in_parent(x))  
  
is_a_missing_or_empty_string(x, .xname = get_name_in_parent(x))  
  
is_a_non_missing_nor_empty_string(x, .xname = get_name_in_parent(x))
```

Arguments

x	A character vector.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname	Not intended to be used directly.

Value

The `is_*` functions return logical vectors for strings which are (non) empty or missing, and the `assert_*` functions throw errors on failure.

Note

In R, `NA_character_` is considered to be a non-empty string (at least by `nzchar`), which is why many functions are needed to clarify the situation.

See Also

[is_character](#), [nzchar](#)

Examples

```
# These functions return a vector:
x <- c("", "a", NA)
is_empty_character(x)
is_non_empty_character(x)
is_missing_or_empty_character(x)
is_non_missing_nor_empty_character(x)

# These functions return a single value:
is_an_empty_string("")
is_an_empty_string("a")
is_an_empty_string(NA_character_)

is_a_non_empty_string("")
is_a_non_empty_string("a")
is_a_non_empty_string(NA_character_)

is_a_missing_or_empty_string("")
is_a_missing_or_empty_string("a")
is_a_missing_or_empty_string(NA_character_)

is_a_non_missing_nor_empty_string("")
is_a_non_missing_nor_empty_string("a")
is_a_non_missing_nor_empty_string(NA_character_)
```

```
assert_all_are_matching_fixed
```

Does the string match a pattern?

Description

Checks to see if the input matches a regular expression or fixed character pattern.

Usage

```
assert_all_are_matching_fixed(x, pattern, opts_fixed = NULL,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_matching_fixed(x, pattern, opts_fixed = NULL,
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))
```

```
assert_all_are_not_matching_fixed(x, pattern, opts_fixed = NULL,  
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))  
  
assert_any_are_not_matching_fixed(x, pattern, opts_fixed = NULL,  
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))  
  
assert_all_are_matching_regex(x, pattern, opts_regex = NULL,  
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))  
  
assert_any_are_matching_regex(x, pattern, opts_regex = NULL,  
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))  
  
assert_all_are_not_matching_regex(x, pattern, opts_regex = NULL,  
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))  
  
assert_any_are_not_matching_regex(x, pattern, opts_regex = NULL,  
  na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))  
  
is_matching_fixed(x, pattern, opts_fixed = NULL,  
  .xname = get_name_in_parent(x))  
  
is_not_matching_fixed(x, pattern, opts_fixed = NULL,  
  .xname = get_name_in_parent(x))  
  
is_matching_regex(x, pattern, opts_regex = NULL,  
  .xname = get_name_in_parent(x))  
  
is_not_matching_regex(x, pattern, opts_regex = NULL,  
  .xname = get_name_in_parent(x))
```

Arguments

x	string
pattern	pattern
opts_fixed	Passed to stri_detect_fixed .
na_ignore	should NAs be ignored or not?
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
opts_regex	Passed to stri_detect_regex .
.xname	Not intended to be used directly.

Author(s)

Aditya Bhagwat

See Also

[stri_detect](#), on which these functions are based.

Examples

```
# Is it safe to eat oysters?
is_matching_fixed(month.name, "r")

# Sometimes it is easier to specify the negative match.
is_matching_regex(LETTERS, "[^AEIOU]")
is_not_matching_regex(LETTERS, "[AEIOU]")

# Matching is vectorized over both x and pattern
(pi_digits <- strsplit(format(pi, digits = 17), "")[[1]])
is_matching_regex(pi_digits, c("[13]", "[59]"))

assert_any_are_matching_regex(pi_digits, c("[13]", "[59]"))

# These checks should fail
assertive.base::dont_stop({
  assert_all_are_matching_regex(pi_digits, c("[13]", "[59]"))
})
```

```
assert_all_are_numeric_strings
```

Does the string contain a number/logical value?

Description

Check to see if a character vector contains numeric/logical strings.

Usage

```
assert_all_are_numeric_strings(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_numeric_strings(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_all_are_logical_strings(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_logical_strings(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

is_numeric_string(x, .xname)

is_logical_string(x, .xname)
```

Arguments

<code>x</code>	A character vector.
<code>na_ignore</code>	A logical value. If FALSE, NA values cause an error; otherwise they do not. Like <code>na.rm</code> in many stats package functions, except that the position of the failing values does not change.
<code>severity</code>	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
<code>.xname</code>	Not intended to be used directly.

Value

`is_numeric_string` returns a logical vector that is TRUE when the string contains numbers. The corresponding `assert_*` functions return nothing but throw an error on failure.

Examples

```
is_numeric_string(c("1", "1.1", "-1.1e1", "one", NA))
# R only treats certain capitalizations of "true" and "false" as logical
x <- c(
  "TRUE", "FALSE", "true", "false", "True", "False", "trUE", "FaLsE",
  "T", "F", "t", "f"
)
is_logical_string(x)

assert_all_are_numeric_strings(c("1", "2.3", "-4.5", "6e7", "8E-9"))
assert_any_are_numeric_strings(c("1", "Not a number"))
```

```
assert_all_are_single_characters
  Is the input a single character?
```

Description

Checks to see if the input is a single character.

Usage

```
assert_all_are_single_characters(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

assert_any_are_single_characters(x, na_ignore = FALSE,
  severity = getOption("assertive.severity", "stop"))

is_single_character(x, .xname)
```

Arguments

x	A character vector.
na_ignore	A logical value. If FALSE, NA values cause an error; otherwise they do not. Like <code>na.rm</code> in many stats package functions, except that the position of the failing values does not change.
severity	How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname	Not intended to be used directly.

Value

`is_single_character` returns TRUE when the input is a single character (as determined by `nchar`; this excludes NA). The `assert_*` functions return nothing but throw an error if the corresponding `is_*` function returns FALSE.

Note

The behaviour of this function has changed so that NA inputs now return NA rather than FALSE.

See Also

[nchar](#)

Examples

```
x <- c("", "a", "aa", NA)
is_single_character(x)
```

character_to_list_of_integer_vectors

Convert a character vector to a list of integer vectors

Description

Split strings by character, then convert to numbers

Usage

```
character_to_list_of_integer_vectors(x)
```

Arguments

x	Input to convert.
---	-------------------

Value

A list of numeric vectors.

See Also

[strsplit](#) and [as.integer](#).

Examples

```
## Not run:
character_to_list_of_integer_vectors(c("123", "4567a"))

## End(Not run)
```

create_regex	<i>Create a regex from components.</i>
--------------	--

Description

Creates a regex from regex components.

Usage

```
create_regex(..., l = list(), sep = "[ - ]?")
```

Arguments

...	Character vectors of regex components.
l	A list of character vectors for alternate specification.
sep	Regex for separating components of complete regex. Defaults to "an optional space or hyphen".

Value

A string containing a regex. Each element in the vectors are pasted together, separated by the sep value. Those character vectors are then preceded by "^" (regex for 'start of string') and followed by "\$" (regex for end of string). Finally, the regexes are collapsed with "|" (regex for 'or').

Examples

```
## Not run:
cas_number_components <- c(
  "[[:digit:]]{1,7}", "[[:digit:]]{2}", "[[:digit:]]"
)
cas_number_rx <- create_regex(rx_components, sep = "-")

## End(Not run)
```

d	<i>Create regex for repeated digits</i>
---	---

Description

Creates a regex string for repeated digits.

Usage

```
d(lo, hi = NA_integer_, optional = FALSE)
```

Arguments

lo	Minimum number of digits to match.
hi	Optional maximum number of digits to match.
optional	If TRUE, the digits are optional.

Value

A character vector of regexes.

Note

If hi is omitted, the returned regex will only match the exact number of digits given by lo.

Examples

```
## Not run:  
d(1:5)  
d(1:5, 6:8)  
d(0:2, Inf)  
  
## End(Not run)
```

matches_regex	<i>Does the input match the regular expression?</i>
---------------	---

Description

Checks that the input matches the regular expression.

Usage

```
matches_regex(x, rx, ignore.case = TRUE, ...)
```

Arguments

<code>x</code>	Input to check.
<code>rx</code>	A regular expression.
<code>ignore.case</code>	Should the case of alphabetic characters be ignored?
<code>...</code>	Passed to grepl .

Value

A logical vector that is TRUE when the input matches the regular expression.

Note

The default for `ignore.case` is different to the default in `grepl`.

See Also

[regex](#) and [regexpr](#).

recycle

Recycle arguments

Description

Explicit recycling of arguments to make them all have the same length.

Usage

```
recycle(...)
```

Arguments

`...` Arguments, usually vectors.

Value

A list of vectors, all with the same length.

Note

The function is based on `rep_len`, which drops attributes (hence this being most appropriate for vector inputs).

See Also

[rep_len](#).

Examples

```
## Not run:
# z is the longest argument, with 6 elements
recycle(x = 1:4, y = list(a = month.abb, b = pi), z = matrix(1:6, nrow = 3))

## End(Not run)
```

strip_invalid_chars *Removes invalid characters from a string.*

Description

Removes invalid characters from a string, leaving only digits.

Usage

```
strip_invalid_chars(x, invalid_chars, char_desc = gettext("invalid"))

strip_non_alphanumeric(x)

strip_non_numeric(x, allow_x = FALSE, allow_plus = FALSE)
```

Arguments

x	Input to strip.
invalid_chars	A regular expression detailing characters to remove.
char_desc	A string describing the characters to remove.
allow_x	If TRUE, the letter "X" is allowed - useful for check digits.
allow_plus	If TRUE, the symbol "+" is allowed - useful for phone numbers.

Value

A character vector of the same length as x, consisting of strings without the characters detailed in the invalid_chars.

Examples

```
## Not run:
strip_invalid_chars(
  " We're floating\tin   space\n\n\n", "[[:space:]]", "whitespace"
)
strip_non_numeric(" +44 800-123-456 ", allow_plus = TRUE)
#Inputs such as factors as coerced to character.
strip_non_alphanumeric(factor(c(" A1\t1AA.", "(B2^2BB)%")))

## End(Not run)
```

Index

as.integer, 9
assert_all_are_empty_character, 2
assert_all_are_logical_strings
 (assert_all_are_numeric_strings),
 6
assert_all_are_matching_fixed, 4
assert_all_are_matching_regex
 (assert_all_are_matching_fixed),
 4
assert_all_are_missing_or_empty_character
 (assert_all_are_empty_character),
 2
assert_all_are_non_empty_character
 (assert_all_are_empty_character),
 2
assert_all_are_non_missing_nor_empty_character
 (assert_all_are_empty_character),
 2
assert_all_are_not_matching_fixed
 (assert_all_are_matching_fixed),
 4
assert_all_are_not_matching_regex
 (assert_all_are_matching_fixed),
 4
assert_all_are_numeric_strings, 6
assert_all_are_single_characters, 7
assert_all_strings_are_not_missing_nor_empty
 (assert_all_are_empty_character),
 2
assert_any_are_empty_character
 (assert_all_are_empty_character),
 2
assert_any_are_logical_strings
 (assert_all_are_numeric_strings),
 6
assert_any_are_matching_fixed
 (assert_all_are_matching_fixed),
 4
assert_any_are_matching_regex
 (assert_all_are_matching_fixed),
 4
assert_any_are_missing_or_empty_character
 (assert_all_are_empty_character),
 2
assert_any_are_non_empty_character
 (assert_all_are_empty_character),
 2
assert_any_are_non_missing_nor_empty_character
 (assert_all_are_empty_character),
 2
assert_any_are_not_matching_fixed
 (assert_all_are_matching_fixed),
 4
assert_any_are_not_matching_regex
 (assert_all_are_matching_fixed),
 4
assert_any_are_numeric_strings
 (assert_all_are_numeric_strings),
 6
assert_any_are_single_characters
 (assert_all_are_single_characters),
 7
assert_any_strings_are_not_missing_nor_empty
 (assert_all_are_empty_character),
 2
assert_is_a_missing_or_empty_string
 (assert_all_are_empty_character),
 2
assert_is_a_non_empty_string
 (assert_all_are_empty_character),
 2
assert_is_a_non_missing_nor_empty_string
 (assert_all_are_empty_character),
 2
assert_is_an_empty_string
 (assert_all_are_empty_character),
 2
character_to_list_of_integer_vectors,

- 8
- create_regex, 9
- d, 10
- grepl, 11
- is_a_missing_or_empty_string
 - (assert_all_are_empty_character), 2
- is_a_non_empty_string
 - (assert_all_are_empty_character), 2
- is_a_non_missing_nor_empty_string
 - (assert_all_are_empty_character), 2
- is_an_empty_string
 - (assert_all_are_empty_character), 2
- is_character, 4
- is_empty_character
 - (assert_all_are_empty_character), 2
- is_logical_string
 - (assert_all_are_numeric_strings), 6
- is_matching_fixed
 - (assert_all_are_matching_fixed), 4
- is_matching_regex
 - (assert_all_are_matching_fixed), 4
- is_missing_or_empty_character
 - (assert_all_are_empty_character), 2
- is_non_empty_character
 - (assert_all_are_empty_character), 2
- is_non_missing_nor_empty_character
 - (assert_all_are_empty_character), 2
- is_not_matching_fixed
 - (assert_all_are_matching_fixed), 4
- is_not_matching_regex
 - (assert_all_are_matching_fixed), 4
- is_not_missing_nor_empty_character
 - (assert_all_are_empty_character), 2
- is_numeric_string
 - (assert_all_are_numeric_strings), 6
- is_single_character
 - (assert_all_are_single_characters), 7
- matches_regex, 10
- nchar, 8
- nzchar, 4
- recycle, 11
- regex, 11
- regexpr, 11
- rep_len, 11
- stri_detect, 6
- stri_detect_fixed, 5
- stri_detect_regex, 5
- strip_invalid_chars, 12
- strip_non_alphanumeric
 - (strip_invalid_chars), 12
- strip_non_numeric
 - (strip_invalid_chars), 12
- strsplit, 9