Package ‘erify’

May 30, 2021

Type Package
Title Check Arguments and Generate Readable Error Messages
Version 0.3.0
Author Renfei Mao
Maintainer Renfei Mao <renfeimao@gmail.com>
Description Provides several validator functions to check if arguments passed by users have valid types, lengths, etc., and if not, to generate informative and good-formatted error messages in a consistent style. Also provides tools for users to create their own validator functions. The error message style used is adopted from <https://style.tidyverse.org/error-messages.html>.
Depends R (>= 4.1.0)
License MIT + file LICENSE
Encoding UTF-8
RoxygenNote 7.1.1
Imports glue, knitr, rstudioapi
Suggests rmarkdown
VignetteBuilder knitr
NeedsCompilation no
Repository CRAN
Date/Publication 2021-05-30 14:40:02 UTC

R topics documented:

back_quote ............................................................... 2
check_binary_classes ............................................... 3
check_bool ............................................................ 4
check_class ............................................................ 6
check_classes ........................................................ 7
Description

Convert an R object to character and add back quotations.

Usage

back_quote(x, recursive = TRUE, as_double = TRUE)

Arguments

x An R object.
recursive Optional. TRUE or FALSE which indicates if to back quote each item of x or to
back quote x as a whole, when x is a vector. The default value is TRUE.
as_double Optional. TRUE or FALSE which indicates if to differentiate between type double
and integer. The default value is TRUE, which means integers are handled as doubles.

Value

A character vector.

Examples

back_quote(1:3)
back_quote(1:3, recursive = FALSE)
back_quote(1:3, as_double = FALSE)
back_quote(NULL)
back_quote(list(c, 1:3, "a"))
check_binary_classes  

Description

Check if the arguments of a binary operator have valid classes, and if not, generate an error message.

Usage

```r
check_binary_classes(
  x,
  y,
  valid_x,
  valid_y = NULL,
  operator = NULL,
  commutative = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  ...
)
```

Arguments

- `x, y` The argument to check, which can be any object.
- `valid_x, valid_y` A character vector which contains the valid classes. `valid_y` is assigned `valid_x`, if not specified.
- `operator` Optional. A single character which represents the binary operator.
- `com mutative` TRUE or FALSE which indicates if arguments `x` and `y` can be swapped around. The default value is TRUE.
- `general` Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.
- `specific` Optional. A single character which gives a detailed description of the error. `glue::glue()` syntax can be used, see "Examples" section. By default, this is generated automatically.
- `supplement` Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see `throw()`. By default, this is left empty.
- `...` Optional. Additional arguments which can be retrieved with `tryCatch()`.

Value

returns an invisible NULL if the argument is valid, or generates an error message.
See Also
"Examples" section in `check_type()` for how to customize error message and how to add and retrieve additional arguments.

vignette("erify") for a gentle introduction to this package.

Examples

```r
## Not run:
x <- 1
class(x) <- c("a", "b")

y <- 2
class(y) <- c("c", "d")

check_binary_classes(x, y, c("d", "e"))
check_binary_classes(x, y, c("d", "e"), operator = "+")

check_binary_classes(x, y, c("d", "e"), c("a", "f"))
check_binary_classes(x, y, c("d", "e"), c("a", "f"), commutative = FALSE)

# customize error message with 'glue::glue()' syntax
check_binary_classes(
x, y, c("d", "e"),
specific = "Left: \{feature_x[1]\}, \{feature_x[2]\}.",
supplement = "Right: \{feature_y[1]\}, \{feature_y[2]\}."
)

## End(Not run)
```

---

**check_bool**  
*Check If Argument Is Single Logical*

**Description**

Check if an argument is TRUE or FALSE, and if not, generate an error message.

**Usage**

```r
check_bool(
x,
name = NULL,
general = NULL,
specific = NULL,
supplement = NULL,
...)
```
**check_bool**

Arguments
---

- `x`: The argument to check, which can be any object.
- `name`: A single character which gives the argument’s name. The name is used in the error message. By default, the name of the argument passed to argument `x` is captured automatically.
- `general`: Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.
- `specific`: Optional. A single character which gives a detailed description of the error. By default, this is generated automatically.
- `supplement`: Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see `throw()`. By default, this is left empty.
- `...`: Optional. Additional arguments which can be retrieved with `tryCatch()`.

**Value**

returns an invisible `NULL` if the argument is valid, or generates an error message.

**See Also**

"Examples" section in `check_type()` for how to customize error message and how to add and retrieve additional arguments.

vignette("erify") for a gentle introduction to this package.

**Examples**

```r
x <- TRUE
cHECK_BOOL(x)
```

```r
# Not run:
# `x` must have type logical
x <- 1
CHECK_BOOL(x)
```

```r
# `x` must have length 1
x <- c(TRUE, FALSE)
CHECK_BOOL(x)
```

```r
# `x` must not be `NA`
x <- NA
CHECK_BOOL(x)
```

```r
# Not run)
```
check_class

_check Argument's Class_

Description

Check if an argument has valid class, and if not, generate an error message.

Usage

\[
\text{check_class}(\text{x}, \text{valid}, \text{name} = \text{NULL}, \text{general} = \text{NULL}, \text{specific} = \text{NULL}, \text{supplement} = \text{NULL}, ...) \]

Arguments

\begin{itemize}
\item \text{x} \quad \text{The argument to check, which can be any object.}
\item \text{valid} \quad \text{A character vector which contains valid classes.}
\item \text{name} \quad \text{A single character which gives the argument's name. The name is used in the error message. By default, the name of the argument passed to argument x is captured automatically.}
\item \text{general} \quad \text{Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.}
\item \text{specific} \quad \text{Optional. A single character which gives a detailed description of the error. glue::glue() syntax can be used, see "Examples" section. By default, this is generated automatically.}
\item \text{supplement} \quad \text{Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see throw(). By default, this is left empty.}
\item \text{...} \quad \text{Optional. Additional arguments which can be retrieved with tryCatch().}
\end{itemize}

Value

returns an invisible NULL if the argument is valid, or generates an error message.

See Also

"Examples" section in check_type() for how to customize error message and how to add and retrieve additional arguments.

vignette("erify") for a gentle introduction to this package.
check_classes

Examples

```r
x <- 1
class(x) <- c("a", "b")

check_class(x, c("a", "c"))

## Not run:
check_class(x, c("c", "d"))

# customize error message with `glue::glue()` syntax
specific <- "Unbelievable! The first class of `{name}` is {feature[1]}."
check_class(x, c("c", "d"), specific = specific)

## End(Not run)
```

check_classes

Check Each Item’s Class

Description

Check if each item of an argument has valid class, and if not, generate an error message.

Usage

```r
check_classes(
  x,
  valid,
  name = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  ...
)
```

Arguments

- **x** - The argument to check, which must be a list.
- **valid** - A character vector which contains valid classes.
- **name** - A single character which gives the argument’s name. The name is used in the error message. By default, the name of the argument passed to argument `x` is captured automatically.
- **general** - Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.
- **specific** - Optional. A single character which gives a detailed description of the error. `glue::glue()` syntax can be used, see “Examples” section. By default, this is generated automatically.
check_content

supplement Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see \code{throw()}. By default, this is left empty.

... Optional. Additional arguments which can be retrieved with \code{tryCatch()}

Value

returns an invisible \code{NULL} if the argument is valid, or generates an error message.

See Also

"Examples" section in \code{check_type()} for how to customize error message and how to add and retrieve additional arguments.

\code{vignette("erify")} for a gentle introduction to this package.

Examples

# argument to check
arg <- lapply(1:10, function(x) {class(x) <- c("a", "b"); x})

check_classes(arg, "a")

## Not run:
check_classes(arg, c("x", "y"))

## End(Not run)

---

\rmarkdown

### check_content

**Check Argument's Content**

Description

Check if an argument is from some given choices or satisfies some requirement, and if not, generate an error message.

Usage

\rmarkdown

```r
check_content(
  x,
  valid,
  name = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  as_double = TRUE,
  ...
)
```


Arguments

- **x**: The argument to check, which can be any object.
- **valid**: can be
  1. a function, which takes `x` as argument and returns `TRUE` or `FALSE`,
  2. an expression, which contains `x` and evaluates to `TRUE` or `FALSE`, or
  3. a string of R code, which evaluates to `TRUE` or `FALSE`, or
  4. a non-empty atomic vector, which contains the valid choices.
- **name**: A single character which gives the argument’s name. The name is used in the error message. By default, the name of the argument passed to argument `x` is captured automatically.
- **general**: Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.
- **specific**: Optional. A single character which gives a detailed description of the error. By default, this is generated automatically.
- **supplement**: Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see `throw()`. By default, this is left empty.
- **as_double**: Optional. `TRUE` or `FALSE` which indicates if to differentiate between type double and integer. The default value is `TRUE`, which means integers are handled as doubles.
- **...**: Optional. Additional arguments which can be retrieved with `tryCatch()`.

Value

returns an invisible `NULL` if the argument is valid, or generates an error message.

See Also

"Examples" section in `check_type()` for how to customize error message and how to add and retrieve additional arguments.

vignette("erify") for a gentle introduction to this package.

Examples

```r
valid <- c(1, 2, 3)

x <- 2L
check_content(x, valid)

## Not run:
# `x` must have the same type with `valid`
x <- "a"
check_content(x, valid)

# `x` must have length 1
x <- c(1, 2)
```
check_content(x, valid)

# differentiate between type double and integer
x <- 2L
check_content(x, valid, as_double = FALSE)

# 'valid' can be a function
ccheck_content(x, is.na, general = "'x' must be 'NA'.")

# 'valid' can be a string of R code
ccheck_content(x, "is.na(x)", general = "'x' must be 'NA'.")

## End(Not run)

---

**check_contents**  
**Check Each Item’s Content**

**Description**
Check if each item of an argument is from some given choices or satisfies some requirement, and if not, generate an error message.

**Usage**
```r
check_contents(
  x,
  valid,
  name = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  as_double = TRUE,
  ...
)
```

**Arguments**
- **x**  
The argument to check, which can be any object.
- **valid**  
can be
  1. a function, which takes x as argument and returns TRUE or FALSE,
  2. an expression, which contains x and evaluates to TRUE or FALSE,
  3. a string of R code, which evaluates to TRUE or FALSE, or
  4. a non-empty atomic vector, which contains the valid choices.
- **name**  
A single character which gives the argument’s name. The name is used in the error message. By default, the name of the argument passed to argument x is captured automatically.
Check Argument's Length

Description

Check if an argument has valid length, and if not, generate an error message.

general
Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.

specific
Optional. A single character which gives a detailed description of the error. By default, this is generated automatically.

supplement
Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see throw(). By default, this is left empty.

as_double
Optional. TRUE or FALSE which indicates if to differentiate between type double and integer. The default value is TRUE, which means integers are handled as doubles.

...
Optional. Additional arguments which can be retrieved with tryCatch().

Value

returns an invisible NULL if the argument is valid, or generates an error message.

See Also

"Examples" section in check_type() for how to customize error message and how to add and retrieve additional arguments.

vignette("erify") for a gentle introduction to this package.

Examples

```r
## Not run:
x <- c(1, 2, 3)
check_contents(x, c(4, 5))

general = "Each item of 'x' must be 'NA'."

# 'valid' can be a function or R code
check_contents(x, is.na, general = general)
check_contents(x, "is.na(x_i)", general = general)

## End(Not run)
```
Usage

check_length(
  x,
  valid,
  name = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  interval = NULL,
  ...
)

Arguments

x
The argument to check, which can be any object.

valid
A numeric vector which contains non-negative integers and NA, used with argument interval to indicate the valid lengths.

name
A single character which gives the argument's name. The name is used in the error message. By default, the name of the argument passed to argument x is captured automatically.

general
Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.

specific
Optional. A single character which gives a detailed description of the error. glue::glue() syntax can be used, see "Examples" section. By default, this is generated automatically.

supplement
Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see throw(). By default, this is left empty.

interval
Optional. TRUE or FALSE which indicates if argument valid is interpreted as an interval or as single lengths. For example, c(1, 10) is interpreted as "larger than 1 and smaller than 10" if interval is TRUE, but as "1 or 10" if FALSE. NA can be used in valid when treated as interval. For example, c(0, NA) means "larger than 0". By default, interval is inferred from valid. For example, if valid has length unequal to 2, it's treated as single lengths.

... Optional. Additional arguments which can be retrieved with tryCatch().

Value

returns an invisible NULL if the argument is valid, or generates an error message.

See Also

"Examples" section in check_type() for how to customize error message and how to add and retrieve additional arguments.

vignette("erify") for a gentle introduction to this package.
Examples

```r
## Not run:
x <- c(1, 2)

# 'valid' as interval
check_length(x, c(1, 3), interval = TRUE)
check_length(x, c(NA, 2))

# 'valid' as single lengths
check_length(x, c(1, 3), interval = FALSE)

# customize error message with `glue::glue()` syntax
specific <- "Oh my god! '{name}'s length is {feature}."
check_length(x, 3, specific = specific)

## End(Not run)
```

check_n  
Check If Argument Is Single Natural Number

Description

Check if an argument is a single natural number, and if not, generate an error message.
Can be used to check indices, for example.

Usage

```r
check_n(
  x,
  name = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  zero = FALSE,
  ...
)

is_n(x, zero = FALSE)
```

Arguments

- **x**: The argument to check, which can be any object.
- **name**: A single character which gives the argument’s name. The name is used in the error message. By default, the name of the argument passed to argument `x` is captured automatically.
- **general**: Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.
specific  Optional. A single character which gives a detailed description of the error. By default, this is generated automatically.

supplement  Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see throw(). By default, this is left empty.

zero  Optional. TRUE or FALSE which indicates if zero is acceptable. The default value is FALSE.

...  Optional. Additional arguments which can be retrieved with tryCatch().

Value

check_n() returns an invisible NULL if the argument is valid, or it generates an error message.
is_n() returns TRUE or FALSE.

See Also

"Examples" section in check_type() for how to customize error message and how to add and retrieve additional arguments.
vignette("erify") for a gentle introduction to this package.

Examples

```r
x <- 1
check_n(x)

x <- 1L
check_n(x)

# Not run:
# \`x\' must be a numeric
x <- \"1\"
check_n(x)

# \`x\' must have length 1
x <- 1:2
check_n(x)

# \`x\' must not be \`NA\'
x <- NA_integer_
check_n(x)

# \`x\' must be larger than 0
x <- -1
check_n(x)

# \`x\' must be an integer in a mathematical sense
x <- 1.1
check_n(x)

# make \`0\' acceptable
```
Check if an argument is a single character and if not, generate an error message. Can be used to check argument names, for example.

### Usage

```r
check_string(
  x,
  name = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  ...
)
```

### Arguments

- **x**: The argument to check, which can be any object.
- **name**: A single character which gives the argument’s name. The name is used in the error message. By default, the name of the argument passed to argument `x` is captured automatically.
- **general**: Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.
- **specific**: Optional. A single character which gives a detailed description of the error. By default, this is generated automatically.
- **supplement**: Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see `throw()`. By default, this is left empty.
- **...**: Optional. Additional arguments which can be retrieved with `tryCatch()`.

### Value

returns an invisible NULL if the argument is valid, or generates an error message.
See Also

"Examples" section in check_type() for how to customize error message and how to add and retrieve additional arguments.

vignette("erify") for a gentle introduction to this package.

Examples

```r
x <- "a"
check_string(x)

## Not run:
#
# `x` must have type character
x <- c
check_string(x)

# `x` must have length 1
x <- c("a", "b")
check_string(x)

# `NA_character_` is not acceptable
x <- NA_character_
check_string(x)

## End(Not run)
```

---

**check_type**  
Check Argument’s Type

Description

Check if an argument has valid type, and if not, generate an error message.

Usage

```r
check_type(
  x,
  valid,
  name = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  ...
)
```
check_type

Arguments

- **x**: The argument to check, which can be any object.
- **valid**: A character vector which contains the valid types.
- **name**: A single character which gives the argument’s name. The name is used in the error message. By default, the name of the argument passed to argument `x` is captured automatically.
- **general**: Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.
- **specific**: Optional. A single character which gives a detailed description of the error. `glue::glue()` syntax can be used, see "Examples" section. By default, this is generated automatically.
- **supplement**: Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see `throw()`. By default, this is left empty.
- **...**: Optional. Additional arguments which can be retrieved with `tryCatch()`.

Value

returns an invisible NULL if the argument is valid, or generates an error message.

See Also

vignette("erify") for a gentle introduction to this package.

Examples

```r
# argument to check
arg <- 10

# returns silently if the argument has valid type
check_type(arg, "double")

## Not run:
check_type(arg, "character")

# specify argument's name
check_type(arg, "character", name = "x")

# specify argument `specific` with `glue::glue()` syntax
specific <- "\{name\}'s type is \(\text{feature}\), which is wrong."
check_type(arg, "character", specific = specific)

# specify argument `supplement`
supplement <- c("You're wrong.", i = "Check your code.")
check_type(arg, "character", supplement = supplement)

# turn off `specific`
options(erify.n = 0)
check_type(arg, "character")
```
## End(Not run)

# add and retrieve additional argument
tryCatch(
  (check_type(arg, "character", your_arg = "your data")),
  error = function(e) e$your_arg
)

---

### check_types

**Check Each Item's Type**

**Description**

Check if each item of an argument has valid type, and if not, generate an error message.

**Usage**

```r
check_types(
  x,
  valid,
  name = NULL,
  general = NULL,
  specific = NULL,
  supplement = NULL,
  ...
)
```

**Arguments**

- `x` The argument to check, which must be a list.
- `valid` A character vector which contains the valid types.
- `name` A single character which gives the argument's name. The name is used in the error message. By default, the name of the argument passed to argument `x` is captured automatically.
- `general` Optional. A single character which is used to give a general statement of the error incurred. By default, this is generated automatically.
- `specific` Optional. A single character which gives a detailed description of the error. `glue::glue()` syntax can be used, see "Examples" section. By default, this is generated automatically.
- `supplement` Optional. A (named) character vector which gives some additional information about the error. The names are used to create bullets, see `throw()`. By default, this is left empty.
- `...` Optional. Additional arguments which can be retrieved with `tryCatch()`.
join

Value

returns an invisible NULL if the argument is valid, or generates an error message.

See Also

"Examples" section in check_type() for how to customize error message and how to add and retrieve additional arguments.
vignette("erify") for a gentle introduction to this package.

Examples

## Not run:
# argument to check
arg <- as.list(1:10)

check_types(arg, "character")

# customize error message with `glue::glue()` syntax
specific <- ""{name}[[i]]" is an {feature}, oh my god!"
check_types(arg, "character", specific = specific)

## End(Not run)

join

Connect Words with Conjunction

Description

Connect given words with a conjunction, e.g. "and" and "or".

Usage

join(words, conjunction = "or")

Arguments

words A vector of list whose items can be converted to characters.

conjunction A single character which represents a conjunction word. The default value is "or".

Value

If has length 1 or less, words is returned. Or items of words are concatenated and returned.

Examples

words <- c("apple", "orange", "Pink Floyd")
join(words, "and")
**throw**  

*Generate and Signal Condition*

**Description**

Generate and signal a condition.

**Usage**

```r
throw(general, specifics = NULL, env = NULL, as = "error", class = NULL, ...)
```

**Arguments**

- **general**: A single character which gives a general statement of the condition.
- **specifics**: Optional. A character vector which gives a list of details of the condition. If is `character(0)`, `throw()` will return silently. If is a named vector, the names are used to create bullets. If the name is "x" or "i", the bullet will be colored and bold. The default name is "x". You can customize bullets with option `erify.bullets`.
- **env**: Optional. An environment or named list which is used to evaluate the R code in the above arguments. See `glue::glue()`.
- **as**: Optional. "error", "warning" or "message" which indicates how to signal the condition. The default value is "error".
- **class**: Optional. A character vector which assigns classes to the condition.
- **...**: Optional. Additional arguments which are stored in the condition and can be retrieved with `tryCatch()`.

**Value**

If `specifics` is `character(0)`, returns an invisible `NULL`. Or signals an error, a warning, or a message.

**Examples**

```r
general <- "You are wrong."

# returns silently
throw(general, character(0))

## Not run:
throw(general)

specifics <- c("Detail 1.", i = "Detail 2.")
throw(general, specifics)

# embed R code with glue syntax
throw("'x' is {x}.", env = list(x = 1))
```
## End(Not run)

# add and retrieve additional argument
tryCatch(
  { throw(general, arg = "I'm an additional argument.") },
  error = function(e) e$arg
)

### Detect Where Code Is Running

#### Description
Check if code is running in RStudio, R Markdown file, R Jupyter Notebook or other contexts. And if is in R Markdown file, check the output format.

#### Usage
where()
is_rmd()
is_rstudio()
is_jupyter()

#### Value
For where():
- If executed in R Markdown file, it returns the output format. If output format is not specified, it returns "rmd".
- If executed in RStudio, it returns "rstudio".
- If executed in R Jupyter Notebook, it returns "jupyter".
- If executed in other contexts, it returns "other".

is_rmd(), is_rstudio() and is_jupyter() return TRUE if executed in their corresponding contexts, or FALSE if not.

#### See Also
rstudioapi::isAvailable(), knitr::pandoc_to().
Index

back_quote, 2

check_binary_classes, 3
check_bool, 4
check_class, 6
check_classes, 7
check_content, 8
check_contents, 10
check_length, 11
check_n, 13
check_string, 15
check_type, 16
check_type(), 4–6, 8, 9, 11, 12, 14, 16, 19
check_types, 18

glue::glue(), 3, 6, 7, 12, 17, 18, 20

is_jupyter (where), 21
is_n(check_n), 13
is_rmd (where), 21
is_rstudio (where), 21

join, 19

knitr::pandoc_to(), 21

rstudioapi::isAvailable(), 21

throw, 20
throw(), 3, 5, 6, 8, 9, 11, 12, 14, 15, 17, 18
tryCatch(), 3, 5, 6, 8, 9, 11, 12, 14, 15, 17, 18, 20

where, 21