Package: svAssert (via r-universe)

May 29, 2026

Type Package

Version 0.1.1

Title 'SciViews::R' - Assertions and Meaningful Error Messages

Description Assertions combining `is_xxx()` and `stop_xxx()` functions for maximum flexibility. Enhanced error messages for 'SciViews::R' based on `cli::cli_abort()` and `rlang::abort()` but also allowing message translation, including for messages in other package, or after the error is thrown.

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 4.4.0)

Imports checkmate (>= 2.3.2), cli (>= 3.6.4), rlang (>= 1.1.1)

Suggests parallel (>= 4.4.0), bench (>= 1.1.4), covr (>= 3.5.0), knitr (>= 1.42), rmarkdown (>= 2.21), spelling (>= 2.2.1), testthat (>= 3.0.0)

License MIT + file LICENSE

URL https://github.com/SciViews/svAssert, https://www.sciviews.org/svAssert/, https://sciviews.r-universe.dev/svAssert

BugReports https://github.com/SciViews/svAssert/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Encoding UTF-8

Language en-US

ByteCompile yes

Config/testthat/edition 3

Config/testthat/parallel true

LazyData true

Repository https://sciviews.r-universe.dev

Date/Publication 2026-03-24 11:21:38 UTC

RemoteUrl https://github.com/SciViews/svAssert

RemoteRef HEAD

RemoteSha 80c9526d3f032b6bd9caa6651874cb6a736cd06a

Contents

`any_infinite` *Check if an object contains infinite values*

Description

Supported are atomic types (see `is.atomic`), lists and data frames.

Usage

```
any_infinite(x)
```

Arguments

`x`	An object to check for the presence of infinite values.

Value

`logical(1)` Returns `TRUE` if any element is `-Inf` or `Inf`.

Examples

```
any_infinite(1:10)
any_infinite(c(1:10, Inf))
data(iris, package = "datasets")
iris[3, 3] <- Inf
any_infinite(iris)
```

---

checkmate_msgs *Messages for Translation: checkmate*

Description

Messages and regular expressions required to identify error strings to translate using `translate()` for 'checkmate'.

Usage

```
checkmate_msgs
```

Format

`checkmate_msgs`**:**
A character vector with messages and corresponding regular expressions as names.

Source

https://github.com/mllg/checkmate (release v2.3.3)

---

general_msgs *Messages for Translation: general*

Description

Messages and regular expressions required to identify error strings to translate using `translate()` for 'general'.

Usage

```
general_msgs
```

Format

`general_msgs`**:**
A character vector with messages and corresponding regular expressions as names.

get_threads *Get or set the number of OpenMP threads used by svAssert*

Description

`get_threads()` returns the current number of threads used for parallel operations. `set_threads()` sets it.

Usage

```
get_threads()

set_threads(n)
```

Arguments

`n`	A positive integer giving the number of threads. Use `1` to disable parallelism.

Value

`get_threads()` returns an integer. `set_threads()` returns the new value invisibly.

Examples

```
(nthreads <- svAssert::get_threads()) # 1 by default
(svAssert::set_threads(parallel::detectCores() - 1L))
# Now svAssert function use parallelism...
# ... your code here...
# Reset it
(svAssert::set_threads(nthreads))
rm(nthreads)
```

is_numeric *Assert that an argument is a vector of type numeric*

Description

Vectors of storage type **integer** and **double** count as **numeric**, c.f. `is.numeric()`. To explicitly assert for integer or double vectors, see `is_integer()`, `is_integerish()` or `is_double()`.

Usage

```
is_numeric(
  x,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  typed.missing = FALSE,
  null.ok = FALSE
)

stop_is_numeric(x, ..., par. = list())
```

Arguments

x	The R object that was tested, typically with is_numeric().
lower	Lower value (number) all elements of x must be greater than or equal to.
upper	Upper value (number) all elements of x must be lower than or equal to.
finite	Logical, indicating whether all elements of x must be finite, default is FALSE.
any.missing	Logical, indicating whether x may contain missing values, default is TRUE.
all.missing	Logical, indicating whether x may be entirely missing values, default is TRUE. An empty vector has no missing values.
len	Expected length of x (integer).
min.len	Minimal length of x (integer).
max.len	Maximal length of x (integer).
unique	Logical, indicating whether all values of x must be unique, default is FALSE.
sorted	Logical, indicating whether all values of x must be sorted in ascending order, default is FALSE.
names	Check for names. Default in NULL (no check). Could be "unnamed" (has no names), "named" (has names), "unique" (has unique names), "strict" (same as unique, but names must be also valid R variable names), or "ids" (same as strict but not enforce uniqueness).
typed.missing	If FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, or NA_character_) and empty vectors are allowed while type-checking atomic input. If TRUE, leads to strict type checking.
null.ok	If set to TRUE, x may also be NULL. In this case only a type check of x is performed, all additional checks are disabled. Default is FALSE.

`...` Any additional arguments (not checked, and not used).

`par.` An optional list with further parameters, like `msg` a custom error message, `arg` the argument name, as a **string**, default is the expression provided to `x`, `mod` a modifier string (only `"!"` is considered here, indicating negation of the condition), `class_id` an identifier to append to the error subclass, `call` the call where the error was generated (the default computes the top-level call of the function(s) that called `error_numeric()` using `stop_top_call()`.

Value

Logical for `is_numeric()`, `TRUE` if `x` passes all checks, `FALSE` otherwise, and the internal `message` option is set with the indication of what failed, can be reused by `error_numeric()` that always create an error message.

Author(s)

Derived from code by Michel Lang, Bernd Bischl, and Dénes Tóth (authors of the {checkmate} package whose code is repackaged here). Documentation is also largely inspired from {checkmate} corresponding documentation.

See Also

`checkmate::check_numeric()`

Examples

```
is_numeric(1.2) # Better using is_num() in this simple case
is_numeric("a")
svAssert::::.checkmate_message() # Get the message set by is_numeric()

my_log <- function(y) {# x must be numeric >= 0
  is_numeric(y, min.len = 1, lower = 0) || stop_is_numeric(y)
  log(y)
}
my_log(1)
# |> try() to catch the error, do not use in real code!
my_log(-1:5) |> try()
```

---

stop_ *Enhanced stop*

Description

`stop_()` is an enhanced version of `stop()` to generate meaningful error with error-recoverable glue interpolation and translation. `error_class()`, `stop_top_call()`, `object_info()` and 'lbl()" are support functions that help building contextual and explicit error messages.

Usage

```
stop_(
  ...,
  domain = NULL,
  class = error_class(call, class_id = class_id),
  class_id = .op$class_id,
  call = NULL,
  parent = NULL,
  .inherit = TRUE,
  .internal = FALSE,
  .file = NULL,
  .envir = parent.frame(),
  .frame = .envir,
  .trace_bottom = NULL,
  .last_call = sys.call(-1L)
)

stop_top_call(nframe = 1L)

error_class(call = parent.frame(), class_id = NULL)

object_info(x)

lbl(expr, width = 30L, nlines = 1L)
```

Arguments

`...`	One or more character strings with the error or warning messages to be translated. Name them `'*' =`, `'i' =`, `'v' =`, `'x' =` or `'!' =` to format a bullet-list with the message items. First message item is considered to use the `'!'` bullet by default. The messages also support glue interpolation and inline markups, see Formatting messages with cli and `cli::format_inline()`.
`domain`	The translation. domain, see `gettext()`. If `NA` or `""`, messages are not translated (use this with messages that are already translated).
`class`	The subclass of the error condition message. By default, it is computed by `error_class()`, using the name of the function in `call` (plus, optionally, the `class_id`), or `"svAssert_error"` by default.
`class_id`	An optional identifier to append to the error subclass.
`call`	The execution environment of a currently running function where the error should be reported from (called the relevant function).
`parent`	Give a condition object when an error is rethrown from a condition handler, such as `withCallingHandlers()` or `rlang::try_fetch()` to chain errors (see Including contextual information with error chains. Indicate `NA` for an unchained rethrow, in case you want to rethrow with a custom error message (do not abuse this, and never hide errors).
`.inherit`	Logical, whether to inherit parent conditions when chaining errors. Default is `TRUE`.

.internal	Logical, whether the error is internal to the package. If TRUE, a footer bullet is added to indicate it to invite the user to report the error to the package authors. Default is FALSE.
.file	A connection or a string where to print the message. The default is context-dependent, see the stdout vs stderr section in [rlang::abort().
.envir	The environment where to evaluate the glue expressions.
.frame	The environment to use for the backtrace (usually the same as .envir).
.trace_bottom	An optional environment to truncate the backtrace in order to display only the most relevant part of it. Default is NULL and it uses callif it is an environment, or .frame otherwise.
.last_call	The last call issued by the user. Different from call when a first argument with dot (.) (e.g., data = (.)) was automatically injected in the call (so-called "data-dot mechanism). In this case, extra information is added to the error message, except if .last_call is NULL (use it to suppress these extra messages).
nframe	The number of frames to go up the call stack to start finding the top call (as soon as .__to_call__. is found in the environment, look in its parent frame).
x	An R object to describe.
expr	An R expression to deparse
width	Maximum width of the deparsed expression
nlines	Maximum number of lines for the deparsed expression

Details

stop_() is a wrapper around rlang::abort() that provides more control on the stop message thanks to cli::cli_abort() glue interpolation and gettext() translation. It can recover from errors in the formatting processes. In this case, it throws the raw error message with a warning. It also adds a class to the error message, with a default one automatically computed by error_class(). It uses stop_top_call() to provide a simple mechanism to point to the execution environment of the running function that is relevant in the context. Add a variable .__top_call__. <- TRUE in the relevant function, or .__top_call__. <- FALSE in an helper function, to be sure to point to the function of interest (most of the time, the function called by the user, see examples). A reminder message inviting to access the backtrace of the error is displayed depending on the svAssert_backtrace_on_error option. Set it to "none" (disable it), "reminder" (default in interactive sessions, show the reminder message), "branch" (display a simplified backtrace) or "full" (default in non-interactive sessions, display the full tree). You rarely need to change the default. It synchronizes with rlang_backtrace_on_error option used in rlang::abort() when needed. You can use rlang::global_entrace() to display classical base R stop() messages in a similar way as stop_() and rlang::abort() do. You can also use rlang::last_error() to revisit the last error message, or rlang::last_trace() to inspect the backtrace of the message. Finally, the display of the error message is customisable. See Customising condition messages.

Value

stop_() is invoked for its side-effects, to stop execution of the current code.

stop_top_call() returns the top call to be used for stop condition messages (to be used as call argument of stop_()). call for stop condition messages.

`error_class()` returns a character string with the error class name computed for `stop_()`, that is, "fun_id_error", or "svAssert_error", by default.

`object_info()` returns a character string describing the R object provided.

`lbl()` returns a deparsed version of an expression suitable for { {lbl(x)}} where <format> could be `.var`, `.arg`, or `.code`.

See Also

`rlang::abort()`, `cli::cli_abort()`, `gettext()` `stop()`

Examples

```
# If you want to include the error messages in the translation strings in
# your package, you have to rename `stop_()` into `stop()` because
# [tools::xgettext2pot()] will only pick up messages in the later ones.
library(svAssert)
stop <- stop_

# Note: the |> try() are there to catch error. Do not use them in your code!
# Correctly formatted stop messages
n <- "some text"
stop("{.var n} must be a numeric vector",
  x = "You've supplied a {.cls {class(n)}} vector.") |> try()

# Incorrectly formatted stop messages (error in glue formatting: missing
# second closing `}` in `{.cls{class(n)}`
stop("{.var n} must be a numeric vector",
  x = "You've supplied a {.cls {class(n)} vector.") |> try()

# Automatic pluralisation
n <- 1:18
stop("{.var n} must be a scalar numeric:",
  i = "There {?is/are} {length(n)} element{?s}.",
  x = "Provide a single numeric, not {object_info(n)}.") |> try()

# When issued from within a function, the function call is indicated
test1 <- function(x) {
  stop("{.var x} must be a scalar numeric:",
    i = "There {?is/are} {length(x)} element{?s} in {.var x}.")
}
test1(1:3) |> try()

# If another function calls `test1()`, error is still reported from test1:

test2 <- function(x) {
  test1(x)
}
test2(1:3) |> try()

# In such a case, it is better to report the error from `test2()`.
# You can do that by stating `._top_call_. <- TRUE` in the body of `test2()`.
test2 <- function(x) {
```

```
  .__top_call__. <- TRUE
  test1(x)
}
test2(1:3) |> try()

# When you design an helper function (a function that is always called from
# another function), you can set `.__top_call__. <- FALSE` to force pointing
# to the calling function. In this case, .__top_call___. <- TRUE is not
# needed in the calling function.
stop_is_scalar_numeric <- function(x) {
  .__top_call__. <- FALSE
  stop("{.var {lbl(substitute(x))}} must be a scalar numeric")
}
test3 <- function(y) {# A function with a proper assertion on y
  (is.numeric(y) && length(y) == 1L) || stop_is_scalar_numeric(y)

  # Do something with y here...
}
test3(1:4) |> try()

# If test3() is called by another function, test4(), focus depends on the
# presence of .__top_call__. in test4()
test4 <- function(y) test3(y)
test4(1:4) |> try()
# or:
test4 <- function(y) {
  .__top_call__. <- TRUE
  test3(y)
}
test4(1:4) |> try()
rm(stop)
```

`stop_equal` *Stop function for equality check*

Description

`stop_equal()` generates an error message for == assertions.

Usage

```
stop_equal(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
```

```
)

`stop_==`(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)
```

Arguments

lhs	The left-hand side of the comparison.
rhs	The right-hand side of the comparison.
...	Additional arguments (not used yet).
mod	A character string indicating the modifier to use for the comparison ("", "any", or "all"). Default is NULL, which is equivalent to "".
na.rm	Logical, indicating whether missing values should be removed before performing the comparison. Default is FALSE.
test_it	Logical, indicating whether the comparison should be actually tested. Default is TRUE. If FALSE, the function only constructs the error message based on the lengths of lhs and rhs, and the presence of missing values.
par.	A list of parameters to customize the error message. Optional fields include msg (message(s) to display at the top), footer(message(s) to add at the bottom), arg (name of first argument), arg2 (name of second argument), id (error class id), mod (the modifier, see mod above), call (the call where the error was generated).

Value

This function always stops with an error.

Examples

```
stop_equal(1, 2) |> try() # A little bit silly with two constants
x <- 1
stop_equal(x, 2) |> try()
x <- 1:2
stop_equal(x, 1) |> try() # Not length 1
stop_equal(x, 1:2, mod = "all") |> try() # Should not call stop_equal() here
stop_equal(x, 1:3, mod = "all") |> try()
stop_equal(x, 1:3, mod = "any") |> try() # Should not call stop_equal() here
stop_equal(x, 2:4, mod = "any")
y <- 2
stop_equal(x, y, mod = "all") |> try()
stop_equal(x, y, mod = "any") |> try() # Should not call stop_equal() here
z <- c(NA, 5, NA)
```

```
stop_equal(x, z, mod = "all") |> try()
stop_equal(x, z, mod = "all", na.rm = TRUE) |> try()
stop_equal(x, z, mod = "any", na.rm = TRUE) |> try()
stop_equal(length(x), 1L) |> try()
stop_equal(NULL, 2L) |> try()
# Extra messages
stop_equal(y, 2, par. = list(msg = c("Some info...", x = "This is wrong"),
  footer = c("*" = "A footer..."))) |> try()
```

`stop_greater_or_equal` *Stop function for greater than or equal (>=) check*

Description

`stop_greater_or_equal()` generates an error message for >= assertions.

Usage

```
stop_greater_or_equal(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)

`stop_>=`(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)
```

Arguments

`lhs`	The left-hand side of the comparison.
`rhs`	The right-hand side of the comparison.
`...`	Additional arguments (not used yet).
`mod`	A character string indicating the modifier to use for the comparison (`""`, `"any"`, or `"all"`). Default is `NULL`, which is equivalent to `""`.
`na.rm`	Logical, indicating whether missing values should be removed before performing the comparison. Default is `FALSE`.

test_it	Logical, indicating whether the comparison should be actually tested. Default is TRUE. If FALSE, the function only constructs the error message based on the lengths of lhs and rhs, and the presence of missing values.
par.	A list of parameters to customize the error message. Optional fields include msg (message(s) to display at the top), footer(message(s) to add at the bottom), arg (name of first argument), arg2 (name of second argument), id (error class id), mod (the modifier, see mod above), call (the call where the error was generated).

Value

This function always stops with an error.

Examples

```
x <- 1
stop_greater_or_equal(x, 3) |> try()
x <- 1:2
# Should not call stop_greater_or_equal() on the following one
stop_greater_or_equal(x, c(0, 3), mod = "any") |> try()
# but yes on the next one
stop_greater_or_equal(x, c(0, 3), mod = "all") |> try()
```

---

stop_greater_than *Stop function for greater than (>) check*

Description

stop_less_than() generates an error message for > assertions.

Usage

```
stop_greater_than(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)

`stop_>`(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
```

```
  par. = list()
)
```

Arguments

lhs	The left-hand side of the comparison.
rhs	The right-hand side of the comparison.
...	Additional arguments (not used yet).
mod	A character string indicating the modifier to use for the comparison ("", "any", or "all"). Default is NULL, which is equivalent to "".
na.rm	Logical, indicating whether missing values should be removed before performing the comparison. Default is FALSE.
test_it	Logical, indicating whether the comparison should be actually tested. Default is TRUE. If FALSE, the function only constructs the error message based on the lengths of lhs and rhs, and the presence of missing values.
par.	A list of parameters to customize the error message. Optional fields include msg (message(s) to display at the top), footer(message(s) to add at the bottom), arg (name of first argument), arg2 (name of second argument), id (error class id), mod (the modifier, see mod above), call (the call where the error was generated).

Value

This function always stops with an error.

Examples

```
x <- 1
stop_greater_than(x, 2) |> try()
x <- 1:2
stop_greater_than(x, 2:3, mod = "any") |> try()
stop_greater_than(x, 2:3, mod = "all") |> try()
```

---

stop_less_or_equal *Stop function for less than or equal (<=) check*

Description

stop_less_or_equal() generates an error message for <= assertions.

Usage

```
stop_less_or_equal(
  lhs,
  rhs,
  ...,
  mod = NULL,
```

```
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)

`stop_<=`(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)
```

Arguments

lhs	The left-hand side of the comparison.
rhs	The right-hand side of the comparison.
...	Additional arguments (not used yet).
mod	A character string indicating the modifier to use for the comparison ("", "any", or "all"). Default is NULL, which is equivalent to "".
na.rm	Logical, indicating whether missing values should be removed before performing the comparison. Default is FALSE.
test_it	Logical, indicating whether the comparison should be actually tested. Default is TRUE. If FALSE, the function only constructs the error message based on the lengths of lhs and rhs, and the presence of missing values.
par.	A list of parameters to customize the error message. Optional fields include msg (message(s) to display at the top), footer(message(s) to add at the bottom), arg (name of first argument), arg2 (name of second argument), id (error class id), mod (the modifier, see mod above), call (the call where the error was generated).

Value

This function always stops with an error.

Examples

```
x <- 1
stop_less_or_equal(x, 0) |> try()
x <- 1:2
# Should not call stop_less_or_equal() in the folowing one
stop_less_or_equal(x, c(0, 3), mod = "any") |> try()
# ...but for the next one
stop_less_or_equal(x, c(0, 3), mod = "all") |> try()
```

stop_less_than *Stop function for less than (<) check*

Description

stop_less_than() generates an error message for < assertions.

Usage

```
stop_less_than(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)

`stop_<`(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)
```

Arguments

lhs	The left-hand side of the comparison.
rhs	The right-hand side of the comparison.
...	Additional arguments (not used yet).
mod	A character string indicating the modifier to use for the comparison ("", "any", or "all"). Default is NULL, which is equivalent to "".
na.rm	Logical, indicating whether missing values should be removed before performing the comparison. Default is FALSE.
test_it	Logical, indicating whether the comparison should be actually tested. Default is TRUE. If FALSE, the function only constructs the error message based on the lengths of lhs and rhs, and the presence of missing values.
par.	A list of parameters to customize the error message. Optional fields include msg (message(s) to display at the top), footer(message(s) to add at the bottom), arg (name of first argument), arg2 (name of second argument), id (error class id), mod (the modifier, see mod above), call (the call where the error was generated).

Value

This function always stops with an error.

Examples

```
x <- 1
stop_less_than(x, 0) |> try()
x <- 1:2
stop_less_than(x, 0:1, mod = "any") |> try()
stop_less_than(x, 0:1, mod = "all") |> try()
```

---

stop_not_equal *Stop function for inequality check*

Description

`stop_not_equal()` generates an error message for `!=` assertions.

Usage

```
stop_not_equal(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)

`stop_!=`(
  lhs,
  rhs,
  ...,
  mod = NULL,
  na.rm = FALSE,
  test_it = TRUE,
  par. = list()
)
```

Arguments

`lhs`	The left-hand side of the comparison.
`rhs`	The right-hand side of the comparison.
`...`	Additional arguments (not used yet).
`mod`	A character string indicating the modifier to use for the comparison (`""`, `"any"`, or `"all"`). Default is `NULL`, which is equivalent to `""`.

na.rm	Logical, indicating whether missing values should be removed before performing the comparison. Default is FALSE.
test_it	Logical, indicating whether the comparison should be actually tested. Default is TRUE. If FALSE, the function only constructs the error message based on the lengths of lhs and rhs, and the presence of missing values.
par.	A list of parameters to customize the error message. Optional fields include msg (message(s) to display at the top), footer(message(s) to add at the bottom), arg (name of first argument), arg2 (name of second argument), id (error class id), mod (the modifier, see mod above), call (the call where the error was generated).

Value

This function always stops with an error.

Examples

```
x <- 1
stop_not_equal(x, 1) |> try()
x <- 1:2
stop_not_equal(x, 1:2, mod = "any") |> try()
stop_not_equal(x, 1:2, mod = "all") |> try()
```

---

stopifnot_ *Assert that expressions are all TRUE, with extended error messages*

Description

If any of the expressions (in ...) are not all TRUE, an error is raised. If a stop_expr() function exists for the expression expr, it is called to generate the error message, otherwise a default message is created.

Usage

```
stopifnot_(...)

get_stop_fun(
  x = NULL,
  expr = substitute(x),
  par. = list(),
  call_it = TRUE,
  force_stop = TRUE
)

mod_not(mod)

mod_content(x, expr)
```

Arguments

`...`	Any number of R expressions, which should each evaluate to (a logical vector of all) TRUE.
`x`	An expression to analyze
`expr`	An expression object
`par.`	A list of additional parameters to pass to the stop function, optionally containing `msg` a string with custom message(s), `arg` a single string with the name of the argument that failed the assertion, `mod` a single string with starting modifier (it is recomputed), `id` an identifier to append to the error class, or `call` the call where the error was generated. Default is an empty list.
`call_it`	If `TRUE`, the corresponding `stop_xxx()` function is called (if it exists) to generate the error message. If `FALSE`, see `force_stop`. Default is `TRUE`.
`force_stop`	If `call_it` is `TRUE` and no `stop_xxx()` function exists for the core expression, or if it exists but does not stop, and if `force_stop = TRUE` (default), a generic error message is generated and the function stops. If `force_stop = FALSE`, no error is raised and information about the stop function is returned.
`mod`	A modifier string (could be `""`, `"!"`, `"any"`, `"all"`, `"!any"`, `"!all"`, or any combination of these).

Details

`get_stop_fun()` and `mod_not()` are utility functions to compute the stop function call and manage a modifier (`!`, `any()` or `all()`) in expressions to ease building error messages.

Value

For `stopifnot_()`, NULL if all statements in ... are TRUE.

For `get_stop_fun(call_it = FALSE)`, a list with `stop_fun` the name of the stop function, `call` the call to the stop function (if it exists, otherwise `NULL`), `mod` the modifier, and `expr` the core expression, excluding the modifier. `mod` can be `NULL` or `""` if no modifier is present. Otherwise, it is "!" (not true), "any" (at least one element is true), "all" (all elements are true), "!any" (all elements are wrong), or "!all" (at least one element is wrong). The error message build by your `stop_xxx()` functions should take this modifier into account to issue the correct error message for the core expression.

For `mod_not()`, `TRUE` if the modifier starts with "!", `FALSE` otherwise.

For `mod_content()`, a message with the content of `expr` and its value `x`.

See Also

`stop_()`, `base::stop()`

Examples

```
# stop <- stop_
# Note that |> try() is just there to catch error; do not use in your code!
x <- 1
```

```
stopifnot_(1 == 1, all.equal(pi, 3.14159265), x < 2) # No error
stopifnot_(1 == 1, all.equal(pi, 3.14159265), x > 2) |> try()
# Compare with the message you got using base::stopifnot():
stopifnot(1 == 1, all.equal(pi, 3.14159265), x > 2) |> try()

stopifnot_(is.character(letters), length(letters) == 1) |> try()
stopifnot_(all.equal(pi, 3.141593), 2 < 2, (1:10 < 12), "a" < "b") |> try()

# get_stop_fun() returns infos about a stop function when call_it = FALSE
get_stop_fun(length(letters) == 1L, call_it = FALSE)
get_stop_fun(length("a") != 1L, call_it = FALSE) # mod == "!", expr = "=="
get_stop_fun(!any(c(TRUE, TRUE, NA)), call_it = FALSE) # mod == "!any"
# all! is the same as !any (and any! is !all)
get_stop_fun(all(!c(TRUE, FALSE, NA)), call_it = FALSE) # mod == "!any"
# Contrived and weird example: it got simplified for mod
get_stop_fun(all(any(!!all(!anyNA(x)))), call_it = FALSE) # mod == "!any"
# Call the stop function (if it exists) to raise the error
get_stop_fun(is.numeric(letters) && length(letters) > 0L) |> try()

# mod_not() can be used exclusively to build either a positive, or a negative
# sentence in your error message when the expression always returns a single
# logical value, because in this case "any" or "all" have no effect.
stop_length_one <- function(x, ..., par. = list()) {
  arg <- lbl(.op$arg %||% par.$arg %||% substitute(x))

  # length(x) == 1 always returns a single logical, can use mod_not() here
  # and safely ignore 'any' or 'all' modifiers
  if (mod_not(par.$mod)) {
    stop(
      "!" = "{.code {arg}} cannot have length 1.")
  } else {
    stop(
      "!" = "{.code {arg}} must have length 1",
      "*" = "Its length is {length(x)}.")
  }
}
length("a") == 1 || stop_length_one("a")
(length(letters) == 1 || stop_length_one(letters)) |> try()
# This avoids writing two stop_xxx() functions, one for == and one for !=
(length("a") != 1 || stop_length_one("a", par. = list(mod = "!"))) |> try()
(!length("a") == 1 || stop_length_one("a", par. = list(mod = "!"))) |> try()
# any() or all() have no effect on single logical and can then be ignored
(any(length("a") != 1) ||
  stop_length_one("a", par. = list(mod = "!all"))) |> try()
rm(stop)
```

`translate` *Translate a message given a set of known messages and regex patterns*

Description

Translate a message given a set of known messages and regex patterns

Usage

```
translate(
  msg,
  dictionary,
  domain = "R-svAssert",
  trim = TRUE,
  format = TRUE,
  ...
)

format_inline_(
  ...,
  .envir = parent.frame(),
  collapse = TRUE,
  keep_whitespace = TRUE,
  post_translate = TRUE
)

create_messages_script(
  dictionary,
  topic,
  pkg_dir = ".",
  export = TRUE,
  source = NULL,
  release = NULL
)
```

Arguments

msg	The message to be translated
dictionary	A character vector with messages to translate and regular expressions to find the corresponding error strings as names.
domain	The domain where to look for messages (see gettext()). The default looks in "R-svAssert", and it should be changed for translations that are available in another package.
trim	Logical, indicating whether leading and trailing whitespace before looking for message translation. TRUE by default.
format	Logical, indicating whether to format the translated message with format_inline_(). Default is TRUE.
...	Messages passed to format_inline().
.envir	The environment in which to evaluate expressions in ....
collapse	Logical, indicating whether to collapse multiple lines into a single string with newlines.
keep_whitespace	Logical, indicating whether to keep whitespace in the formatted message.
post_translate	Logical, indicating whether to translate ", and " and ", or " in the final message.

topic	The identifier for a dictionary. Usually, it should be the name of the package that generates the messages, but it is not mandatory. Use "general" for a set with no particular topic.
pkg_dir	The root directory of the sources of an R package. The produced messages-topic.R script file is created in the R/ subdirectory.
export	Logical, indicating whether the created object should be exported. Default is TRUE.
source	An optional URL indicating where the messages come from, e.g., a GitHub repository.
release	An optional string indicating the release version of the source where the messages were extracted from.

Value

For translate() the translated message if a correspondence is found in the dictionary and a translation is known, or the original message otherwise.

For format_inline_() a formatted message as a string. If an error occurs during formatting, a warning is issued and a simple concatenation of the arguments is returned instead.

For create_messages_script() the name of the script file that is created is returned invisibly. The function is used for its side effect of creating the adequate script file.

See Also

cli::format_inline()

Examples

```
checkmate_msgs[1:3] # First 3 {checkmate} messages
msg <- "Unknown class identifier 'my_class'"
translate(msg, checkmate_msgs, domain = "R-svAssert")
```

Index