Package: arg (via r-universe)

May 26, 2026

Type Package

Title Clean and Simple Argument Checking

Version 0.1.1

Description Checks function arguments, ideally for use in R packages. Uses a simple interface and produces clean, informative error messages using 'cli'.

Depends R (>= 4.1.0)

Imports rlang (>= 1.1.0), cli (>= 3.6.4), stats, utils

License GPL (>=2)

Encoding UTF-8

URL https://ngreifer.github.io/arg/, https://github.com/ngreifer/arg

BugReports https://github.com/ngreifer/arg/issues

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-05-26 20:33:22 UTC

RemoteUrl https://github.com/ngreifer/arg

RemoteRef HEAD

RemoteSha b614e99ec86bafc7286a7b911cf958eccce05c85

Contents

`arg_between` *Check Values in Range*

Description

Checks whether values are within a range (`arg_between()`), greater than some value (`arg_gt()`), greater than or equal to some value (`arg_gte()`), less than some value (`arg_lt()`), or less than or equal to some value (`arg_lte()`).

Usage

```
arg_between(
  x,
  range = c(0, 1),
  inclusive = TRUE,
  .arg = rlang::caller_arg(x),
  .msg = NULL,
  .call
)

arg_gt(x, bound = 0, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_gte(x, bound = 0, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_lt(x, bound = 0, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_lte(x, bound = 0, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

x	the argument to be checked
range	a vector of two values identifying the required range.
inclusive	a logical vector identifying whether the range boundaries are inclusive or not. If only one value is supplied, it is applied to both range boundaries. Default is TRUE, which will not throw an error if x is equal to the boundaries.
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.
bound	the bound to check against. Default is 0.

Details

x is not checked for type, as it is possible for values other than numeric values to be passed and compared; however, an error will be thrown if typeof(x) is not equal to typeof(range) or typeof(bound). NA values of x will be removed. The arguments to range, inclusive, and bound are checked for appropriateness.

Value

Returns NULL invisibly if an error is not thrown.

Examples

```
z <- 2

try(arg_between(z, c(1, 3))) # No error
try(arg_between(z, c(1, 2))) # No error
try(arg_between(z, c(1, 2),
                inclusive = FALSE)) # Error
try(arg_between(z, c(1, 2),
                inclusive = c(TRUE, FALSE))) # Error

try(arg_gt(z, 0))  # No error
try(arg_gt(z, 2))  # Error
try(arg_gte(z, 2)) # No error

try(arg_lt(z, 0))  # Error
try(arg_lt(z, 2))  # Error
try(arg_lte(z, 2)) # No error

try(arg_lte(z, "3")) # Error: wrong type
```

arg_character *Check Character Argument*

Description

Checks whether an argument is a character vector (`arg_character()`), a character scalar (`arg_string()`), or a factor (`arg_factor()`).

Usage

```
arg_character(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_string(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_factor(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

`x`	the argument to be checked
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.
`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

`NA` values in `arg_string()` will cause an error to be thrown.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`is.character()`, `is.factor()`, `rlang::is_string()`

Examples

```
f <- function(z) {
  arg_string(z)
}

try(f("a")) # No error
try(f(c("a", "b"))) # Error: arg_string() requires scalar
try(f(NA)) # NAs not allowed for arg_string()
```

`arg_data` *Check Common Argument Types*

Description

Checks whether an argument is an atomic vector (`arg_atomic()`), a dimensionless atomic vector (`arg_vector()`), a `list` (`arg_list()`), a data frame (`arg_data.frame()`), a `matrix` (`arg_matrix()`), an `array` (`arg_array()`), a rectangular data set (`arg_data()`), or an `environment` (`arg_env()`).

Usage

```
arg_atomic(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_vector(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_list(x, df_ok = FALSE, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_data.frame(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_matrix(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_array(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_data(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_env(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

`x`	the argument to be checked
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.

`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.
`df_ok`	`logical`; whether to allow data frames (which are technically lists). Default is `FALSE` to throw an error on a data frame input.

Details

Atomic vectors are checked using `is.atomic()`. Because matrices are considered atomic vectors, `arg_vector()` additionally checks that there is no `"dims"` attribute, i.e., that `length(dim(x)) == 0L`. `arg_data()` checks whether `x` is either a data frame or a matrix. `arg_list()` checks whether `x` is a list; when `df = FALSE`, it throws an error when `x` is a data frame, even though data frames are lists.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`is.atomic()`, `is.list()`, `is.data.frame()`, `is.matrix()`, `is.array()`, `is.environment()`, `arg_is()`

Examples

```
vec <- 1:6
mat <- as.matrix(vec)
dat <- as.data.frame(mat)
lis <- as.list(vec)
nul <- NULL

# arg_atomic
try(arg_atomic(vec))
try(arg_atomic(mat))
try(arg_atomic(dat))
try(arg_atomic(lis))
try(arg_atomic(nul))

# arg_vector
try(arg_vector(vec))
try(arg_vector(mat))

# arg_matrix
try(arg_matrix(vec))
try(arg_matrix(mat))
try(arg_matrix(dat))

# arg_data.frame
try(arg_data.frame(vec))
```

```
try(arg_data.frame(mat))
try(arg_data.frame(dat))

# arg_data
try(arg_data(vec))
try(arg_data(mat))
try(arg_data(dat))
```

`arg_dots_supplied` *Check Supplied Dots Argument*

Description

Checks whether an argument was supplied to `...`.

Usage

```
arg_dots_supplied(..., .msg = NULL, .call)

arg_dots_not_supplied(..., .msg = NULL, .call)
```

Arguments

`...`	the `...` argument passed from a calling function. The argument is not evaluated. See Examples.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.
`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

`arg_dots_supplied()` checks whether arguments were passed to the `...` (i.e., "dots") argument of a function. These arguments are not evaluated. `arg_dots_supplied()` throws an error if `...length()` is 0. It can be useful for when a function must have additional arguments. For example, `arg_or()` and `when_supplied()` use `arg_dots_supplied()`.

`arg_dots_not_supplied()` checks whether no arguments were passed to `...`, again without evaluating the arguments and using `...length()`. It can be useful when a function appears to allow further arguments (e.g., because it is a method of a generic), but when the author does not want the user to supply them.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

arg_supplied()

Examples

```
f <- function(...) {
  arg_dots_supplied(...)
}

try(f(1)) # No error: argument supplied
try(f())  # Error!
```

---

arg_element *Check Element*

Description

Checks whether values in an argument are all elements of some set (arg_element()) or are not all elements of a set (arg_not_element()). arg_element() throws an error when all(is.element(x, values)) is FALSE, and arg_not_element() throws an error when any(is.element(x, values)) is TRUE.

Usage

```
arg_element(x, values, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_not_element(x, values, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

x	the argument to be checked
values	the values to check x against.
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

arg_element() can be useful for checking whether an argument matches one or more allowed values. It's important that a check is done beforehand to ensure x is the correct type, e.g., using arg_string() or arg_character() if values is a character vector. No partial matching is used; the values must match exactly. Use match_arg() to allow partial matching and return the full version of the supplied argument.

arg_not_element() can be useful for ensuring there is no duplication of values between an argument and some other set.

Value

Returns NULL invisibly if an error is not thrown.

See Also

is.element(), match_arg()

Examples

```
f <- function(z) {
  arg_element(z, c("opt1", "opt2", "opt3"))
}

try(f("opt1"))            # No error
try(f(c("opt1", "opt2"))) # No error, all are elements
try(f(c("opt1", "opt1"))) # No error: repeats allowed
try(f("bad_arg"))         # Error: not an element of set
try(f("opt"))             # Error: partial matching not allowed
try(f(c("opt1", "bad_arg"))) # Error: one non-match

g <- function(z) {
  arg_not_element(z, c("bad1", "bad2", "bad3"))
}

try(g("bad1"))            # Error: z is an element
try(g(c("bad1", "opt2"))) # Error: at least one bad match
try(g("opt1"))            # No error: not an element
try(g(c("opt1", "opt2"))) # No error, none are elements
```

---

arg_equal *Check Equal Arguments*

Description

Checks whether two arguments are equal (arg_equal()) or not equal (arg_not_equal()).

Usage

```
arg_equal(
  x,
  x2,
  ...,
  .arg = rlang::caller_arg(x),
  .arg2 = rlang::caller_arg(x2),
  .msg = NULL,
  .call
)

arg_not_equal(
  x,
  x2,
  ...,
  .arg = rlang::caller_arg(x),
  .arg2 = rlang::caller_arg(x2),
  .msg = NULL,
  .call
)
```

Arguments

x, x2	the objects to compare with each other.
...	other arguments passed to all.equal().
.arg, .arg2	the names of the objects being compared; used in the error message if triggered. Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

Tests for equality are performed by evaluating whether isTRUE(all.equal(x, x2, ...)) is TRUE or FALSE.

Value

Returns NULL invisibly if an error is not thrown.

See Also

all.equal()

Examples

```
f <- function(x, y, ...) {
  arg_equal(x, y, ...)
}

try(f(x = 1, y = 1))      ## No error
try(f(x = 1L, y = 1.0))   ## No error despite type difference
try(f(x = 1, y = 1.00001, ## No error, within tolerance
      tolerance = .001))

try(f(x = 1, y = 2))       ## Error: different
try(f(x = 1, y = 1.00001)) ## Error

g <- function(x, y, ...) {
  arg_not_equal(x, y, ...)
}

try(g(x = 1, y = 1)) ## Error
try(g(x = 1, y = 2)) ## No error
```

---

arg_formula *Check Formula Argument*

Description

Checks whether an argument is a formula.

Usage

```
arg_formula(
  x,
  one_sided = NULL,
  .arg = rlang::caller_arg(x),
  .msg = NULL,
  .call
)
```

Arguments

x	the argument to be checked
one_sided	NULL or logical; if TRUE, checks that x is a formula with only one side (the right side); if FALSE, checks that x is a formula with both sides; if NULL (the default), checks only that x is a formula.
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.

`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`rlang::is_formula()`, `arg_is()`

Examples

```
form1 <- ~a + b
form2 <- y ~ a + b
not_form <- 1:3

try(arg_formula(form1))    # No error
try(arg_formula(form2))    # No error
try(arg_formula(not_form)) # Error: not a formula

try(arg_formula(form1,
                one_sided = TRUE)) # No error
try(arg_formula(form2,
                one_sided = TRUE)) # Error, not one-sided

try(arg_formula(form1,
                one_sided = FALSE)) # Error, only one-sided
try(arg_formula(form2,
                one_sided = FALSE)) # No error
```

---

arg_function *Check Function Argument*

Description

Checks whether an argument is a function.

Usage

```
arg_function(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

x	the argument to be checked
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Value

Returns NULL invisibly if an error is not thrown.

See Also

rlang::is_function(), arg_is()

Examples

```
f <- function(z) {
  arg_function(z)
}

try(f(print))   # No error
try(f("print")) # Error: must be a function
```

---

arg_index *Check Index Argument*

Description

Checks whether an argument is a valid column index (arg_index()) of a data set or a vector thereof (arg_indices()).

Usage

```
arg_index(
  x,
  data,
  .arg = rlang::caller_arg(x),
  .arg_data = rlang::caller_arg(data),
  .msg = NULL,
  .call
```

```
)

arg_indices(
  x,
  data,
  .arg = rlang::caller_arg(x),
  .arg_data = rlang::caller_arg(data),
  .msg = NULL,
  .call
)
```

Arguments

x	the argument to be checked
data	a data set (i.e., a matrix or data frame)
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.arg_data	the name of the argument supplied to data to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

For arg_indices(), an error will be thrown unless one of the following are true:

- x is a vector of counts (see arg_counts()) less than or equal to ncol(data)
- x is a character vector with values a subset of colnames(data)

For arg_index(), x additionally must have length equal to 1. Passing arg_index() ensures that data[, x] (if data is a matrix) or data[[x]] (if x is a data frame) evaluate correctly.

If data has no column names, an error will be thrown if x is a character vector.

Value

Returns NULL invisibly if an error is not thrown.

See Also

arg_counts(), arg_character()

Examples

```
dat <- data.frame(col1 = 1:5,
                  col2 = 6:10)

f <- function(z) {
  arg_index(z, dat)
}

try(f(1))         # No error
try(f(3))         # Error: not a valid index
try(f("col1"))    # No error
try(f("bad_col")) # Error: not a valid index
try(f(1:2))       # Error: arg_index() requires scalar

mat <- matrix(1:9, ncol = 3)

g <- function(z) {
  arg_indices(z, mat)
}

try(g(1))     # No error
try(g(1:3))   # No error
try(g("col")) # Error: `mat` has no names
```

---

arg_is *Check Argument Class*

Description

Checks whether an argument is a of a specified class (`arg_is()`) or is not of a specified class (`arg_is_not()`).

Usage

```
arg_is(x, class, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_is_not(x, class, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

`x`	the argument to be checked
`class`	a character vector of one or more classes to check `x` against.
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.

`.call` the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

`arg_is()` and `arg_is_not()` use `inherits()` to test for class membership. `arg_is()` throws an error only if no elements of `class(x)` are in `class`. `arg_is_not()` throws an error if any elements of `class(x)` are in `class`.

Sometimes this can be too permissive; combining `arg_is()` with `arg_and()` can be useful for requiring that an object is of multiple classes. Alternatively, combining `arg_is_not()` with `arg_or()` can be useful for requiring that an object is not a specific combination of classes. See Examples.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`inherits()`, `arg_or()`; `arg_data()` for testing for specific kinds of objects (vectors, matrices, data frames, and lists)

Examples

```
obj <- structure(list(1),
                 class = "test")

try(arg_is(obj, "test"))            # No error
try(arg_is(obj, c("test", "quiz"))) # No error
try(arg_is(obj, "quiz"))            # Error

try(arg_is_not(obj, "test"))            # Error
try(arg_is_not(obj, c("test", "quiz"))) # Error
try(arg_is_not(obj, "quiz"))            # No error

# Multiple classes
obj2 <- structure(list(1),
                  class = c("test", "essay"))

try(arg_is(obj2, c("test", "quiz")))     # No error
try(arg_is_not(obj2, c("test", "quiz"))) # Error

## Require argument to be of multiple classes
try(arg_and(obj2,
            arg_is("test"),
            arg_is("quiz")))

## Require argument to not be a specific combination of
## multiple classes
```

```
try(arg_or(obj2,
           arg_is_not("test"),
           arg_is_not("essay")))
```

`arg_length` *Check Argument Length*

Description

Checks whether an argument has a specified length.

Usage

```
arg_length(x, len = 1L, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

`x`	the argument to be checked
`len`	`integer`; the allowed length(s) of `x`. Default is 1.
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.
`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

`len` can contain multiple allowed counts; an error will be thrown only if `length(x)` is not equal to any value of `len`.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`length()`, `arg_non_null()` to specifically test that an object's length is or is not 0.

Examples

```
obj <- 1:4

try(arg_length(obj, 1))
try(arg_length(obj, 4))
try(arg_length(obj, c(1, 4)))

# These test the same thing:
try(arg_length(obj, c(0:3)))
try(when_not_null(obj,
                  arg_length(1:3)))
```

`arg_logical` *Check Logical Argument*

Description

Checks whether an argument is a logical vector (`arg_logical()`) or a logical scalar (`arg_flag()`), i.e., a single logical value. Logical values include `TRUE` and `FALSE`.

Usage

```
arg_logical(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_flag(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

`x`	the argument to be checked
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.
`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

`NA` values in `arg_flag()` will cause an error to be thrown.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`is.logical()`, `rlang::is_bool()`

Examples

```
obj <- TRUE

try(arg_flag(obj))    # No error
try(arg_logical(obj)) # No error

obj <- c(TRUE, FALSE)

try(arg_flag(obj))    # Error: must be a scalar
try(arg_logical(obj)) # No error

obj <- 1L

try(arg_flag(obj))    # Error must be logical
try(arg_logical(obj)) # Error must be logical
```

---

arg_named *Check Named Argument*

Description

Checks whether an argument has valid (non-NULL, non-empty, and non-NA) names.

Usage

```
arg_named(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_colnamed(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

x	the argument to be checked
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

arg_named() works for vectors, lists, and data frames. To check whether a matrix has column names, use arg_colnamed() (which also works for data frames, but not vectors or lists).

Value

Returns NULL invisibly if an error is not thrown.

See Also

rlang::is_named(), names(), colnames(), arg_data()

Examples

```
obj <- c(1,
         B = 2,
         C = 3)

try(arg_named(obj)) # Error: one name is blank

names(obj)[1L] <- "A"
try(arg_named(obj)) # No error

obj2 <- unname(obj)
try(arg_named(obj2)) # Error: no names

# Matrix and data frame
mat <- matrix(1:6, ncol = 2L)
colnames(mat) <- c("A", "B")

try(arg_named(mat))    # Error: matrices are not named
try(arg_colnamed(mat)) # No error

dat <- as.data.frame(mat)
try(arg_named(dat))    # No error: data frames are named
try(arg_colnamed(dat)) # No error

colnames(mat) <- NULL
try(arg_colnamed(mat)) # Error: no colnames
```

---

arg_no_NA *Check NA in Argument*

Description

Checks whether an argument does not contain any NA values (arg_no_NA()), contains only NA values (arg_all_NA()), or is a scalar NA (arg_is_NA()).

Usage

```
arg_no_NA(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_is_NA(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_all_NA(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

x	the argument to be checked
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

arg_no_NA() throws an error when anyNA(x) is 0. arg_all_NA() throws an error when all(is.na(x)) is not FALSE. arg_is_NA() throws an error when length(x) is not 1 or anyNA(x) is FALSE.

arg_no_NA() is useful for checking that a meaningful argument was supplied. arg_all_NA() and arg_is_NA() are primarily used for in arg_or() to denote that NA is an allowed argument.

Value

Returns NULL invisibly if an error is not thrown.

See Also

arg_non_null(), arg_supplied(), anyNA()

Examples

```
f <- function(x) {
  arg_no_NA(x)  ## x must not be NA
}

try(f(1))           ## No error
try(f(NA))          ## Error: x is NA
try(f(c(1, NA, 3))) ## Error: x contains NA

f2 <- function(y) {
  arg_all_NA(y) ## y must be NA
}
```

```
try(f2(NA))          ## No error
try(f2(c(NA, NA)))   ## No error
try(f2(1))           ## Error: y is not NA
try(f2(c(1, NA, 3))) ## Error: y is not all NA
```

`arg_non_null` *Check NULL Argument*

Description

Checks whether an argument is non-`NULL` (`arg_non_null()`) or is `NULL` (`arg_null()`). `arg_non_null()` throws an error when `length(x)` is `0`. `arg_null()` throws an error when `length(x)` is not `0`.

Usage

```
arg_non_null(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_null(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

`x`	the argument to be checked
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.
`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

Here, `NULL` refers to any length-0 object, including `NULL`, `logical(0L)`, `list()`, `3[FALSE]`, etc. `arg_non_null()` is useful for checking that a meaningful argument was supplied. `arg_null()` is primarily used for in `arg_or()` to denote that `NULL` is an allowed argument.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`arg_length()`, `arg_no_NA()`, `arg_supplied()`

Examples

```
f <- function(x = NULL, y = NULL) {
  arg_non_null(x) ## x must not be NULL
  arg_null(y)     ## y must be NULL
}

try(f(x = 1,    y = NULL)) ## No error
try(f(x = NULL, y = NULL)) ## Error: x is NULL
try(f(x = 1,    y = 1))    ## Error: y is non-NULL

# Any object of length 0 is considered NULL
try(f(x = numeric())) ## Error: x is NULL
try(f(x = list()))    ## Error: x is NULL

test <- c(1, 2)[c(FALSE, FALSE)]
try(f(x = test))      ## Error: x is NULL

# arg_null() is best used in and_or():
f2 <- function(z) {
  arg_or(z,
         arg_null(),
         arg_number())
}

try(f2(NULL)) ## No error; z can be NULL
try(f2(1))    ## No error; z can be a number
try(f2(TRUE)) ## Error: z must be NULL or a number
```

---

arg_numeric *Check Numeric Argument*

Description

Checks whether an argument is numeric, with some additional constraints if desired. `arg_numeric()` simply checks whether the argument is numeric. `arg_number()` checks whether the argument is a numeric scalar (i.e., a single number). `arg_whole_numeric()` and `arg_whole_number()` check whether the argument is a whole numeric vector or scalar, respectively. `arg_counts()` and `arg_count()` check whether the argument is a non-negative whole numeric vector or scalar, respectively.

Usage

```
arg_numeric(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_number(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_whole_numeric(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_whole_number(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

```
arg_counts(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_count(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

x	the argument to be checked
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

A whole number is decided by testing whether the value is an integer (i.e., using is.integer()) or if abs(x - trunc(x)) < sqrt(.Machine$double.eps). This is the same tolerance used by all.equal() to compare values.

Value

Returns NULL invisibly if an error is not thrown.

See Also

is.integer(), rlang::is_integerish()

Examples

```
count <- 3
whole <- -4
num <- .943

try(arg_number(count))       # No error
try(arg_whole_number(count)) # No error
try(arg_count(count))        # No error

try(arg_number(whole))       # No error
try(arg_whole_number(whole)) # No error
try(arg_count(whole))        # Error: negatives not allowed

try(arg_number(num))       # No error
try(arg_whole_number(num)) # Error: not a whole number
try(arg_count(num))        # Error: not a count
```

```
nums <- c(0, .5, 1)

try(arg_number(nums))  # Error: not a single number
try(arg_numeric(nums)) # No error
try(arg_counts(nums))  # Error: not counts
```

---

arg_or *Check That Arguments Meet Some or All Criteria*

Description

`arg_or()` checks whether an argument meets at least one given criterion. `arg_and()` checks whether an argument meets all given criteria.

Usage

```
arg_or(x, ..., .arg = rlang::caller_arg(x), .msg = NULL, .call)

arg_and(x, ..., .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

`x`	the argument to be checked
`...`	`arg_*()` functions or unevaluated function calls to be applied to `x`. See Examples.
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.
`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

For `arg_or()`, an error will be thrown only if `x` fails all of the supplied checks. This can be useful when an argument can take on one of several input types. For `arg_and()`, and error will be thrown if `x` fails any of the supplied checks. This is less useful on its own, as the checks can simply occur in sequence without `arg_and()`, but `arg_and()` can be supplied to the `...` argument of `arg_or()` to create more complicated criteria.

The `...` arguments can be passed either as functions, e.g.,

```
arg_or(x,
       arg_number,
       arg_string,
       arg_flag)
```

or as unevaluated function calls with the `x` argument absent, e.g.,

```
arg_or(x,
       arg_number(),
       arg_string(),
       arg_flag())
```

or as a mixture of both.

These functions do their best to provide a clean error message composed of all the error messages for the failed checks. With many options, this can yield a complicated error message, so use caution. `arg_and()` marks with a check (`v`) any passed checks and with a cross (`x`) any failed checks.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`arg_supplied()`, `when_not_null()`

Examples

```
# `arg_or()`
f <- function(z) {
  arg_or(z,
         arg_number,
         arg_string,
         arg_flag)
}

try(f(1))      # No error
try(f("test")) # No error
try(f(TRUE))   # No error
try(f(1:4))    # Error: neither a number, string,
#              #        or flag, but a vector

# `arg_and()`
g <- function(z) {
  arg_and(z,
          arg_counts,
          arg_length(len = 2),
          arg_lt(bound = 5))
}

try(g(c(1, 2)))     # No error
try(g(c(1, 7)))     # Error: not < 5
```

```
try(g(c(1.1, 2.1))) # Error: not counts
try(g(4))           # Error: not length 2
try(g("bad"))       # Error: no criteria satisfied

# Chaining together `arg_and()` and `arg_or()`
h <- function(z) {
  arg_or(z,
         arg_all_NA,
         arg_and(arg_count,
                 arg_lt(5)),
         arg_and(arg_string,
                 arg_element(c("a", "b", "c"))))
}

try(h(NA))  # No error
try(h(1))   # No error
try(h("a")) # No error
try(h(7))   # Error: not < 5
try(h("d")) # Error: not in "a", "b", or "c"
```

`arg_supplied` *Check Supplied Argument*

Description

Checks whether an argument with no default value was supplied. An error will be thrown if `rlang::is_missing(x)` is `TRUE`, i.e., if an argument with no default is omitted from a function call.

Usage

```
arg_supplied(x, .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

`x`	the argument to be checked
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.
`.msg`	an optional alternative message to display if an error is thrown instead of the default message.
`.call`	the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Value

Returns NULL invisibly if an error is not thrown.

See Also

arg_non_null(), arg_no_NA(), rlang::is_missing(), rlang::check_required()

Examples

```
f <- function(z) {
  arg_supplied(z)
}

try(f(1)) ## No error: argument supplied
try(f())  ## Error!

# Will not throw for NULL or default arguments
try(f(NULL)) ## No error: argument supplied

f2 <- function(z = NULL) {
  arg_supplied(z)
}

try(f2()) ## No error; default provided
```

---

arg_symmetric *Check Symmetric Matrix Argument*

Description

arg_symmetric() checks whether an argument is a square, symmetric, numeric matrix. arg_cov() checks whether an argument is like a covariance matrix (i.e., square and symmetric with non-negative diagonal entries). arg_cor() checks whether an argument is like a correlation matrix (i.e., square and symmetric with all values between -1 and 1 and ones on the diagonal). arg_distance() checks whether an argument is like a distance matrix (i.e., square and symmetric with non-negative entries and zeros on the diagonal).

Usage

```
arg_symmetric(
  x,
  tol = 100 * .Machine$double.eps,
  ...,
  .arg = rlang::caller_arg(x),
  .msg = NULL,
  .call
)
```

```
arg_cov(
  x,
  tol = 100 * .Machine$double.eps,
  ...,
  .arg = rlang::caller_arg(x),
  .msg = NULL,
  .call
)

arg_cor(
  x,
  tol = 100 * .Machine$double.eps,
  ...,
  .arg = rlang::caller_arg(x),
  .msg = NULL,
  .call
)

arg_distance(
  x,
  tol = 100 * .Machine$double.eps,
  ...,
  .arg = rlang::caller_arg(x),
  .msg = NULL,
  .call
)
```

Arguments

x	the argument to be checked
tol	numeric; the tolerance value used to assess symmetry and any numerical bounds. Passed to isSymmetric().
...	other arguments passed to isSymmetric() (and eventually to all.equal()).
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

These functions check that an argument is a square, symmetric, numeric matrix. This can be useful when a function can accept a covariance matrix, correlation matrix, or distance matrix. arg_cov()

will throw an error if any values on its diagonal are less than `-tol`. `arg_cor()` will throw an error if any of its values are greater than `1 + tol` in absolute value or if the diagonal entries are not between `-1 - tol` and `1 + tol`. `arg_distance()` will thrown an error if any of its values are less than `-tol` or if the diagonal entries are not between `-tol` and `tol`. The tolerance is just slightly greater than 0 to allow for numeric imprecision.

No checks on semi-definiteness or rank are performed.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`arg_numeric()`, `arg_matrix()`, `arg_between()`, `isSymmetric()`

Examples

```
set.seed(1234)
mat <- matrix(rnorm(12), ncol = 3L) # Non square
sym_mat <- -crossprod(mat)          # Square, symmetric
cov_mat <- cov(mat)                 # Covariance
cor_mat <- cor(mat)                 # Correlation
dist_mat <- as.matrix(dist(mat))    # Distance

try(arg_symmetric(mat))     # Error: not square
try(arg_symmetric(sym_mat)) # No error

try(arg_cov(sym_mat)) # Error: diagonal must be non-negative
try(arg_cov(cov_mat)) # No error

try(arg_cor(cov_mat)) # Error: values must be in [-1, 1]
try(arg_cor(cor_mat)) # No error

try(arg_distance(cor_mat))  # Error: diagonal must be 0
try(arg_distance(dist_mat)) # No error
```

---

arg_unique *Check Unique Argument*

Description

Checks whether an argument contains only unique values.

Usage

```
arg_unique(x, ..., .arg = rlang::caller_arg(x), .msg = NULL, .call)
```

Arguments

x	the argument to be checked
...	further argument passed to anyDuplicated(), which performs the checking.
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.msg	an optional alternative message to display if an error is thrown instead of the default message.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Value

Returns NULL invisibly if an error is not thrown.

See Also

anyDuplicated()

Examples

```
f <- function(z) {
  arg_unique(z)
}

try(f(1:3))     # No error
try(f(NULL))    # No error for NULL
try(f(c(1, 1))) # Error: repeated values
```

---

err *Throw an Error, Warning, or Message*

Description

These functions are similar to stop()/cli::cli_abort(), warning()/cli::cli_warn(), and message()/cli::cli_inform(), throwing an error, warning, and message, respectively. Minor processing is done to capitalize the first letter of the message, add a period at the end (if it makes sense to), and add information about the calling function.

Usage

```
err(m, .call, .envir = rlang::caller_env(), ...)

wrn(m, immediate = TRUE, .call, .envir = rlang::caller_env(), ...)

msg(m, .call = NULL, .envir = rlang::caller_env(), ...)
```

Arguments

m	the message to be displayed, passed to the message argument of rlang::abort(), rlang::warn(), or rlang::inform().
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. See the call argument of rlang::abort() for details. Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.
.envir	the environment to evaluate the glue expressions in. See rlang::abort() for details. Typically this does not need to be changed.
...	other arguments passed to rlang::abort(), rlang::warn(), or rlang::inform().
immediate	whether to output the warning immediately (TRUE, the default) or save all warnings until the end of execution (FALSE). See warning() for details. Note that the default here differs from that of warning().

Details

These functions are simple wrappers for the corresponding functions in **rlang**, namely rlang::abort() for err(), rlang::warn() for wrn(), and rlang::inform() for msg(), but which function almost identically to the **cli** versions. Their main differences are that they additionally process the input (capitalizing the first character of the message and adding a period to the end if needed, unless multiple strings are provided). err() is used inside all arg_*() functions in **arg**.

Value

err() throws an error condition. wrn() throws a warning condition and invisibly returns the formatted warning message as a string. msg() signals a message and invisibly returns NULL.

See Also

- Base versions: stop(), warning(), message()
- **rlang** versions: rlang::abort(), rlang::warn(), rlang::inform()
- **cli** versions: cli::cli_abort(), cli::cli_warn(), cli::cli_inform()

Examples

```
f <- function(x) {
  err("this is an error, and {.arg x} is {.type {x}}")
}
```

```
try(f(1))

g <- function(x) {
  wrn("this warning displayed last", immediate = FALSE)
  wrn("this warning displayed first")
}

try(g(1))

h <- function() {
  msg("is a period added at the end?")
  msg("not when the message ends in punctuation!")
  msg(c("or when multiple",
        "!" = "messages",
        "v" = "are",
        "*" = "displayed"))
  msg("otherwise yes")
}

h()
```

`match_arg` *Argument Verification*

Description

An alternative to `match.arg()` with improved error messages via **cli**. Returns the first choice if `x` is `NULL`, and supports partial matching.

Usage

```
match_arg(
  x,
  choices,
  several.ok = FALSE,
  ignore.case = TRUE,
  .context = NULL,
  .arg = rlang::caller_arg(x),
  .call
)
```

Arguments

`x`	a string (or character vector if `several.ok = TRUE`) to match against `choices`. If `NULL`, the first element of `choices` is returned.
`choices`	a character vector of valid values.
`several.ok`	`logical`; if `TRUE`, `x` may contain more than one element. Defaults to `FALSE`.

ignore.case	logical; if FALSE, the matching is case sensitive, and if TRUE (the default), case is ignored.
.context	an optional character string providing context for error messages (e.g., a function or object name). Prepended to the error message when supplied before being evaluated by **cli**.
.arg	the name of the argument supplied to x to appear in error messages. The default is to extract the argument's name using rlang::caller_arg(). Ignored if .msg is supplied.
.call	the execution environment of a currently running function, e.g. .call = rlang::current_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to err(). Set to NULL to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

Partial matching is supported via pmatch(). If no match is found, an error is thrown listing the valid choices.

Checks are run on x prior to matching: first, arg_supplied() is used to check whether x was supplied; then arg_string() (if several.ok = TRUE) or arg_character() (if several.ok = FALSE) is used to check whether x is a valid string or character vector, respectively.

When ignore.case = TRUE (the default), an initial case-sensitive match is run, and if any values in x are unmatched, a second, case-insensitive match is run. When ignore.case = FALSE, only the first match is run. This ensures that exact matches (on both content and case) are prioritized before case-insensitive matches.

Value

A character string (or vector, if several.ok = TRUE) of the matched element(s) from choices. If there is no match, an error is thrown. When several.ok = TRUE, no error is thrown if there is at least one match.

See Also

- match.arg() for the base R version
- pmatch() for the function implementing partial string matching
- arg_element() for a version without type checking and that doesn't support partial matching
- rlang::arg_match() for a similar **rlang** function that doesn't support partial matching.

Examples

```
f <- function(z = NULL) {
  match_arg(z, c("None", "Exact", "Partial"),
            ignore.case = TRUE)
}

try(f())            # "None" (first choice returned)
```

```
try(f("partial"))   # "Partial"
try(f("p"))         # "Partial" (partial match)
try(f(c("e", "p"))) # Error: several.ok = FALSE

# several.ok = TRUE
g <- function(z = NULL) {
  match_arg(z, c("None", "Exact", "Partial"),
            several.ok = TRUE)
}

try(g("exact"))               # Error: case not ignored
try(g("Exact"))               # "Exact"
try(g(c("Exact", "Partial"))) # "Exact", "Partial"
try(g(c("Exact", "Wrong")))   # "Exact"
try(g(c("Wrong1", "Wrong2"))) # Error: no match

h <- function(z = NULL) {
  match_arg(z, c("None", "Exact", "Partial"),
            .context = "in {.fun h},")
}

try(h("Wrong")) # Error with context
```

`when_supplied` *Check Arguments When Supplied*

Description

These functions check arguments only when they are supplied (`when_supplied()`) or when not `NULL` (`when_not_null()`). Multiple checks can be applied in sequence. This allows arguments not to have to be supplied, but checks them only if they are.

Usage

```
when_supplied(x, ..., .arg = rlang::caller_arg(x), .call)

when_not_null(x, ..., .arg = rlang::caller_arg(x), .call)
```

Arguments

`x`	the argument to be checked
`...`	`arg_*()` functions or unevaluated function calls to be applied to `x`. See Examples.
`.arg`	the name of the argument supplied to `x` to appear in error messages. The default is to extract the argument's name using `rlang::caller_arg()`. Ignored if `.msg` is supplied.

`.call` the execution environment of a currently running function, e.g. `.call = rlang::current_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. Passed to `err()`. Set to `NULL` to omit call information. The default is to search along the call stack for the first user-facing function in another package, if any.

Details

An error will be thrown only if `x` is supplied and fails one of the supplied checks (`when_supplied()`) or is not `NULL` and fails one of the supplied checks (`when_not_null()`).

The `...` arguments can be passed either as functions, e.g.,

```
when_supplied(x,
              arg_number,
              arg_gt)
```

or as unevaluated function calls with the `x` argument absent, e.g.,

```
when_supplied(x,
              arg_number(),
              arg_gt(bound = 0))
```

or as a mixture of both.

`when_supplied()` only makes sense to use for an argument that has no default value but which can be omitted. `when_not_null()` makes sense to use for an argument with a default value of `NULL`.

Value

Returns `NULL` invisibly if an error is not thrown.

See Also

`arg_or()`, `arg_supplied()`

Examples

```
f <- function(z) {
  when_supplied(z,
                arg_number,
                arg_between(c(0, 1)))
}

try(f())    # No error: not supplied
try(f("a")) # Error: not a number
try(f(2))   # Error: not within 0-1 range
try(f(.7))  # No error: number within range

g <- function(z = NULL) {
  when_not_null(z,
```

```
                arg_number,
                arg_between(c(0, 1)))
}

try(g())     # No error: NULL okay
try(g(NULL)) # No error: NULL okay
try(g("a"))  # Error: not a number
try(g(2))    # Error: not within 0-1 range
try(g(.7))   # No error: number within range
```

Index