Package ‘checkmate’

February 6, 2020

Type Package

Title Fast and Versatile Argument Checks

Description Tests and assertions to perform frequent argument checks. A substantial part of the package was written in C to minimize any worries about execution time overhead.

Version 2.0.0

URL https://github.com/mllg/checkmate

URLNote https://github.com/mllg/checkmate

BugReports https://github.com/mllg/checkmate/issues

NeedsCompilation yes

ByteCompile yes

Encoding UTF-8

Depends R (>= 3.0.0)

Imports backports (>= 1.1.0), utils

Suggests R6, fastmatch, data.table (>= 1.9.8), devtools, ggplot2, knitr, magrittr, microbenchmark, rmarkdown, testthat (>= 0.11.0), tinytest (>= 1.1.0), tibble

License BSD_3_clause + file LICENSE

VignetteBuilder knitr

RoxygenNote 7.0.2

R topics documented:

checkmate-package ............................................. 3
allMissing .......................................................... 6
anyInfinite .......................................................... 7
anyNaN ............................................................... 8
asInteger ............................................................ 8
assert ................................................................. 11
AssertCollection .................................................. 12
checkAccess ......................................................... 13
checkArray ........................................................ 14
checkAtomic ........................................................ 17
checkAtomicVector ................................................ 20
checkCharacter ..................................................... 23
checkChoice ........................................................ 27
checkClass .......................................................... 29
checkComplex ....................................................... 32
checkCount .......................................................... 35
checkDataFrame ..................................................... 38
checkDataTable ...................................................... 42
checkDate ............................................................ 46
checkDirectoryExists ............................................. 50
checkDisjunct ....................................................... 52
checkDouble ........................................................ 53
checkEnvironment ............................................... 57
checkFactor .......................................................... 59
checkFALSE .......................................................... 64
checkFileExists .................................................... 65
checkFlag ............................................................. 67
checkFormula ........................................................ 68
checkmate-package

Description

Tests and assertions to perform frequent argument checks. A substantial part of the package was written in C to minimize any worries about execution time overhead.
Check scalars
  • checkFlag
  • checkCount
  • checkNumber
  • checkInt
  • checkString
  • checkScalar
  • checkScalarNA

Check vectors
  • checkLogical
  • checkNumeric
  • checkDouble
  • checkInteger
  • checkIntegerish
  • checkCharacter
  • checkComplex
  • checkFactor
  • checkList
  • checkPOSIXct
  • checkVector
  • checkAtomic
  • checkAtomicVector
  • checkRaw

Check attributes
  • checkClass
  • checkMultiClass
  • checkNames
  • checkNamed (deprecated)

Check compound types
  • checkArray
  • checkDataFrame
  • checkMatrix
Check other built-in R types

- `checkDate`
- `checkEnvironment`
- `checkFunction`
- `checkFormula`
- `checkNull`

Check sets

- `checkChoice`
- `checkSubset`
- `checkSetEqual`
- `checkDisjunct`

File IO

- `checkFileExists`
- `checkDirectoryExists`
- `checkPathForOutput`
- `checkAccess`

Popular data types of third party packages

- `checkDataTable`
- `checkR6`
- `checkTibble`

Safe coercion to integer

- `asCount`
- `asInt`
- `asInteger`

Quick argument checks using a DSL

- `qassert`
- `qassertr`

Misc

- `checkOS` (check operating system)
- `assert` (combine multiple checks into an assertion)
- `anyMissing`
- `allMissing`
- `anyNaN`
- `wf` (which.first and which.last)
allMissing

Check if an object contains missing values

Description

Supported are atomic types (see `is.atomic`), lists and data frames. Missingness is defined as `NA` or `NaN` for atomic types and data frame columns, `NULL` is defined as missing for lists.

`allMissing` applied to a `data.frame` returns `TRUE` if at least one column has only non-missing values. If you want to perform the less frequent check that there is not a single non-missing observation present in the `data.frame`, use `all(sapply(df, allMissing))` instead.

Usage

```r
allMissing(x)
anyMissing(x)
```

Arguments

- `x` [ANY]
  
  Object to check.

Value

`logical(1)` Returns `TRUE` if any (`anyMissing`) or all (`allMissing`) elements of `x` are missing (see details), `FALSE` otherwise.
anyInfinite

Examples

allMissing(1:2)
allMissing(c(1, NA))
allMissing(c(NA, NA))
x = data.frame(a = 1:2, b = NA)
# Note how allMissing combines the results for data frames:
allMissing(x)
all(sapply(x, allMissing))
anyMissing(c(1, 1))
anyMissing(c(1, NA))
anyMissing(list(1, NULL))

x = iris
x[, "Species"] = NA
anyMissing(x)
allMissing(x)

anyInfinite Check if an object contains infinite values

Description

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

Usage

anyInfinite(x)

Arguments

x [ANY]
Object to check.

Value

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

Examples

anyInfinite(1:10)
anyInfinite(c(1:10, Inf))
iris[3, 3] = Inf
anyInfinite(iris)
anyNaN

Check if an object contains NaN values

Description

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

Usage

anyNaN(x)

Arguments

x

[ANY]
Object to check.

Value

logical(1) Returns TRUE if any element is NaN.

Examples

anyNaN(1:10)
anyNaN(c(1:10, NaN))
iris[3, 3] = NaN
anyNaN(iris)

asInteger

Convert an argument to an integer

Description

asInteger is intended to be used for vectors while asInt is a specialization for scalar integers and asCount for scalar non-negative integers. Convertible are (a) atomic vectors with all elements NA and (b) double vectors with all elements being within tol range of an integer.

Note that these functions may be deprecated in the future. Instead, it is advised to use `assertCount`, `assertInt` or `assertIntegerish` with argument `coerce` set to TRUE instead.
Usage

```r
asInteger(
  x,
  tol = sqrt(.Machine$double.eps),
  lower = -Inf,
  upper = Inf,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  .var.name = vname(x)
)
```

```r
asCount(
  x,
  na.ok = FALSE,
  positive = FALSE,
  tol = sqrt(.Machine$double.eps),
  .var.name = vname(x)
)
```

```r
asInt(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  tol = sqrt(.Machine$double.eps),
  .var.name = vname(x)
)
```

Arguments

- **x** [any]
  Object to convert.
- **tol** [double(1)]
  Numerical tolerance used to check whether a double or complex can be converted. Default is `sqrt(.Machine$double.eps)`.
- **lower** [numeric(1)]
  Lower value all elements of `x` must be greater than or equal to.
- **upper** [numeric(1)]
  Upper value all elements of `x` must be lower than or equal to.
- **any.missing** [logical(1)]
  Are vectors with missing values allowed? Default is `TRUE`. 

all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

unique [logical(1)]
Must all values be unique? Default is FALSE.

sorted [logical(1)]
Elements must be sorted in ascending order. Missing values are ignored.

names [character(1)]
Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.

.var.name [character(1)]
Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.

na.ok [logical(1)]
Are missing values allowed? Default is FALSE.

positive [logical(1)]
Must x be positive (>= 1)? Default is FALSE.

Details
This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value
Converted x.

Examples
asInteger(c(1, 2, 3))
asCount(1)
asInt(1)
assert

Combine multiple checks into one assertion

Description

You can call this function with an arbitrary number of check* functions, i.e. functions provided by this package or your own functions which return TRUE on success and the error message as character(1) otherwise. The resulting assertion is successful, if combine is "or" (default) and at least one check evaluates to TRUE or combine is "and" and all checks evaluate to TRUE. Otherwise, assert throws an informative error message.

Usage

assert(..., combine = "or", .var.name = NULL)

Arguments

... [any]
List of calls to check functions.

combine [character(1)]
"or" or "and" to combine the check functions with an OR or AND, respectively.

.var.name [character(1)]
Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.

Value

Throws an error if all checks fail and invisibly returns TRUE otherwise.

Examples

```r
x = 1:10
assert(checkNull(x), checkInteger(x, any.missing = FALSE))
## Not run:
x = 1
assert(checkChoice(x, c("a", "b")), checkDataFrame(x))
## End(Not run)
```
AssertCollection

Collect multiple assertions

Description

The function `makeAssertCollection()` returns a simple stack-like closure you can pass to all functions of the `assert*`-family. All messages get collected and can be reported with `reportAssertions()`. Alternatively, you can easily write your own report function or customize the output of the report function to a certain degree. See the example on how to push custom messages or retrieve all stored messages.

Usage

```r
makeAssertCollection()

reportAssertions(collection)
```

Arguments

- `collection` [AssertCollection]
  Object of type “AssertCollection” (constructed via `makeAssertCollection`).

Value

`makeAssertCollection()` returns an object of class “AssertCollection” and `reportCollection` returns invisibly `TRUE` if no error is thrown (i.e., no message was collected).

Examples

```r
x = "a"
coll = makeAssertCollection()

coll$isEmpty()
assertNumeric(x, add = coll)
coll$isEmpty()
coll$push("Custom error message")
coll$getMessage()

## Not run:
  reportAssertions(coll)

## End(Not run)
```
checkAccess

Check file system access rights

Description

Check file system access rights

Usage

checkAccess(x, access = "")
check_access(x, access = "")
assertAccess(x, access = "", .var.name = vname(x), add = NULL)
assert_access(x, access = "", .var.name = vname(x), add = NULL)
testAccess(x, access = "")
test_access(x, access = "")
expect_access(x, access = "", info = NULL, label = vname(x))

Arguments

x [any]
Object to check.

access [character(1)]
Single string containing possible characters ‘r’, ‘w’ and ‘x’ to force a check for read, write or execute access rights, respectively. Write and executable rights are not checked on Windows.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.
checkArray

Value

Depending on the function prefix: If the check is successful, the functions assertAccess/assert_access return x invisibly, whereas checkAccess/check_access and testAccess/test_access return TRUE. If the check is not successful, assertAccess/assert_access throws an error message, testAccess/test_access returns FALSE, and checkAccess returns a string with the error message. The function expect_access always returns an expectation.

See Also

Other filesystem: checkDirectoryExists(), checkFileExists(), checkPathForOutput()

Examples

# Is R's home directory readable?
testAccess(R.home(), "r")

# Is R's home directory writeable?
testAccess(R.home(), "w")

Description

Check if an argument is an array

Usage

checkArray(
x,
  mode = NULL,
  any.missing = TRUE,
  d = NULL,
  min.d = NULL,
  max.d = NULL,
  null.ok = FALSE
)

check_array(
x,
  mode = NULL,
  any.missing = TRUE,
  d = NULL,
  min.d = NULL,
  max.d = NULL,
  null.ok = FALSE
)
checkArray

```r
assertArray(
  x,
  mode = NULL,
  any.missing = TRUE,
  d = NULL,
  min.d = NULL,
  max.d = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)
```

```r
assert_array(
  x,
  mode = NULL,
  any.missing = TRUE,
  d = NULL,
  min.d = NULL,
  max.d = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)
```

```r
testArray(
  x,
  mode = NULL,
  any.missing = TRUE,
  d = NULL,
  min.d = NULL,
  max.d = NULL,
  null.ok = FALSE
)
```

```r
test_array(
  x,
  mode = NULL,
  any.missing = TRUE,
  d = NULL,
  min.d = NULL,
  max.d = NULL,
  null.ok = FALSE
)
```

```r
expect_array(
  x,
  mode = NULL,
```
checkArray

    any.missing = TRUE,
    d = NULL,
    min.d = NULL,
    max.d = NULL,
    null.ok = FALSE,
    info = NULL,
    label = vname(x)
  )

Arguments

  x     [any]  Object to check.
  mode  [character(1)]  Storage mode of the array. Arrays can hold vectors, i.e. “logical”, “integer”, “integerish”, “double”, “numeric”, “complex”, “character” and “list”. You can also specify “atomic” here to explicitly prohibit lists. Default is NULL (no check).
  any.missing [logical(1)]  Are missing values allowed? Default is TRUE.
  d  [integer(1)]  Exact number of dimensions of array x. Default is NULL (no check).
  min.d [integer(1)]  Minimum number of dimensions of array x. Default is NULL (no check).
  max.d [integer(1)]  Maximum number of dimensions of array x. Default is NULL (no check).
  null.ok [logical(1)]  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.
  .var.name [character(1)]  Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
  add [AssertCollection]  Collection to store assertion messages. See AssertCollection.
  info [character(1)]  Extra information to be included in the message for the testthat reporter. See expect_that.
  label [character(1)]  Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertArray/assert_array return x invisibly, whereas checkArray/check_array and testArray/test_array return TRUE. If the check is not successful, assertArray/assert_array throws an error message, testArray/test_array returns FALSE, and checkArray returns a string with the error message. The function expect_array always returns an expectation.
checkAtomic

See Also
Other basetypes: checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()
Other compound: checkDataFrame(), checkDataTable(), checkMatrix(), checkTibble()

Examples

checkArray(array(1:27, dim = c(3, 3, 3)), d = 3)

checkAtomic x, any.missing = TRUE, all.missing = TRUE, len = NULL, min.len = NULL, max.len = NULL, unique = FALSE, names = NULL

check_atomic x, any.missing = TRUE, all.missing = TRUE, len = NULL, min.len = NULL, max.len = NULL, unique = FALSE, names = NULL

assertAtomic x, any.missing = TRUE, all.missing = TRUE,

Description
For the definition of “atomic”, see is.atomic.

Usage

checkAtomic(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL
)

check_atomic(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL
)

assertAtomic(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
checkAtomic

```r
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
names = NULL,
.var.name = vname(x),
add = NULL
)

assert_atomci(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  .var.name = vname(x),
  add = NULL
)

testAtomic(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL
)

test_atomic(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL
)

expect_atomic(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL
)
checkAtomic

len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
names = NULL,
info = NULL,
label = vname(x)
)

Arguments

x [any]
Object to check.

any.missing [logical(1)]
Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

unique [logical(1)]
Must all values be unique? Default is FALSE.

names [character(1)]
Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.

.var.name [character(1)]
Name of the checked object to print in assertions.Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertAtomic/assert_atomic return x invisibly, whereas checkAtomic/check_atomic and testAtomic/test_atomic return TRUE. If the check is not successful, assertAtomic/assert_atomic throws an error message, testAtomic/test_atomic
returns FALSE, and checkAtomic returns a string with the error message. The function expect_atomic always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Other atomicvector: checkAtomicVector(), checkVector()

Examples

testAtomic(letters, min.len = 1L, any.missing = FALSE)

checkAtomicVector x

Check that an argument is an atomic vector

Description

An atomic vector is defined slightly different from specifications in is.atomic and is.vector: An atomic vector is either logical, integer, numeric, complex, character or raw and can have any attributes except a dimension attribute (like matrices). I.e., a factor is an atomic vector, but a matrix or NULL are not. In short, this is basically equivalent to is.atomic(x) && !is.null(x) && is.null(dim(x)).

Usage

checkAtomicVector(
  x,
  any.missing = TRUE, all.missing = TRUE, len = NULL, min.len = NULL, max.len = NULL, unique = FALSE, names = NULL
)

check_atomic_vector(
  x,
  any.missing = TRUE, all.missing = TRUE, len = NULL, min.len = NULL, max.len = NULL, unique = FALSE,
check AtomicVector

names = NULL

assertAtomicVector(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    names = NULL,
    .var.name = vname(x),
    add = NULL
)

assert_atomic_vector(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    names = NULL,
    .var.name = vname(x),
    add = NULL
)

testAtomicVector(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    names = NULL
)

test_atomic_vector(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
names = NULL
)

expect_atomic_vector(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  info = NULL,
  label = vname(x)
)

Arguments

x [any]
Object to check.

any.missing [logical(1)]
Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

unique [logical(1)]
Must all values be unique? Default is FALSE.

names [character(1)]
Check for names. See checkNamed for possible values. Default is “any” which performs no check at all.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.
checkCharacter

Value

Depending on the function prefix: If the check is successful, the functions assertAtomicVector/assert_atomic_vector return x invisibly, whereas checkAtomicVector/check_atomic_vector and testAtomicVector/test_atomic_vector return TRUE. If the check is not successful, assertAtomicVector/assert_atomic_vector throws an error message, testAtomicVector/test_atomic_vector returns FALSE, and checkAtomicVector returns a string with the error message. The function expect_atomic_vector always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomic(), checkCharacter(), checkComplex(), checkDataframe(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXx(), checkRaw(), checkVector()

Other atomicvector: checkAtomic(), checkVector()

Examples

testAtomicVector(letters, min.len = 1L, any.missing = FALSE)

checkCharacter x invisibly, whereas checkAtomicVector/check_atomic_vector and testAtomicVector/test_atomic_vector return TRUE. If the check is not successful, assertAtomicVector/assert_atomic_vector throws an error message, testAtomicVector/test_atomic_vector returns FALSE, and checkAtomicVector returns a string with the error message. The function expect_atomic_vector always returns an expectation.

Check if an argument is a vector of type character

checkCharacter x invisibly, whereas checkAtomicVector/check_atomic_vector and testAtomicVector/test_atomic_vector return TRUE. If the check is not successful, assertAtomicVector/assert_atomic_vector throws an error message, testAtomicVector/test_atomic_vector returns FALSE, and checkAtomicVector returns a string with the error message. The function expect_atomic_vector always returns an expectation.

Check if an argument is a vector of type character

Usage

clickCharacter x invisibly, whereas checkAtomicVector/check_atomic_vector and testAtomicVector/test_atomic_vector return TRUE. If the check is not successful, assertAtomicVector/assert_atomic_vector throws an error message, testAtomicVector/test_atomic_vector returns FALSE, and checkAtomicVector returns a string with the error message. The function expect_atomic_vector always returns an expectation.

Check if an argument is a vector of type character

Usage

clickCharacter x invisibly, whereas checkAtomicVector/check_atomic_vector and testAtomicVector/test_atomic_vector return TRUE. If the check is not successful, assertAtomicVector/assert_atomic_vector throws an error message, testAtomicVector/test_atomic_vector returns FALSE, and checkAtomicVector returns a string with the error message. The function expect_atomic_vector always returns an expectation.

Check if an argument is a vector of type character

Usage

clickCharacter x invisibly, whereas checkAtomicVector/check_atomic_vector and testAtomicVector/test_atomic_vector return TRUE. If the check is not successful, assertAtomicVector/assert_atomic_vector throws an error message, testAtomicVector/test_atomic_vector returns FALSE, and checkAtomicVector returns a string with the error message. The function expect_atomic_vector always returns an expectation.

Check if an argument is a vector of type character

Usage

clickCharacter x invisibly, whereas checkAtomicVector/check_atomic_vector and testAtomicVector/test_atomic_vector return TRUE. If the check is not successful, assertAtomicVector/assert_atomic_vector throws an error message, testAtomicVector/test_atomic_vector returns FALSE, and checkAtomicVector returns a string with the error message. The function expect_atomic_vector always returns an expectation.

Check if an argument is a vector of type character

Usage
checkCharacter

x,
min.chars = NULL,
pattern = NULL,
fixed = NULL,
ignore.case = FALSE,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
names = NULL,
null.ok = FALSE
)

assertCharacter(
x,
min.chars = NULL,
pattern = NULL,
fixed = NULL,
ignore.case = FALSE,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
names = NULL,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

assert_character(
x,
min.chars = NULL,
pattern = NULL,
fixed = NULL,
ignore.case = FALSE,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
names = NULL,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)
names = NULL,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

testCharacter(
  x,
  min.chars = NULL,
  pattern = NULL,
  fixed = NULL,
  ignore.case = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  null.ok = FALSE
)

test_character(
  x,
  min.chars = NULL,
  pattern = NULL,
  fixed = NULL,
  ignore.case = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  null.ok = FALSE
)

expect_character(
  x,
  min.chars = NULL,
  pattern = NULL,
  fixed = NULL,
  ignore.case = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
Arguments

x [any]
Object to check.

min.chars [integer(1)]
Minimum number of characters for each element of x.

pattern [character(1L)]
Regular expression as used in `grepl`. All non-missing elements of x must comply to this pattern.

fixed [character(1)]
Substring to detect in x. Will be used as pattern in `grepl` with option `fixed` set to TRUE. All non-missing elements of x must contain this substring.

ignore.case [logical(1)]
See `grepl`. Default is FALSE.

any.missing [logical(1)]
Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

unique [logical(1)]
Must all values be unique? Default is FALSE.

sorted [logical(1)]
Elements must be sorted in ascending order. Missing values are ignored.

names [character(1)]
Check for names. See `checkNamed` for possible values. Default is “any” which performs no check at all. Note that you can use `checkSubset` to check for a specific set of names.

null.ok [logical(1)]
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.
checkChoice

Description

Check if an object is an element of a given set

Details

This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value

Depending on the function prefix: If the check is successful, the functions assertCharacter/assert_character return x invisibly, whereas checkCharacter/check_character and testCharacter/test_character return TRUE. If the check is not successful, assertCharacter/assert_character throws an error message, testCharacter/test_character returns FALSE, and checkCharacter returns a string with the error message. The function expect_character always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

testCharacter(letters, min.len = 1, any.missing = FALSE)
testCharacter(letters, min.chars = 2)
testCharacter("example", pattern = "xa")
Usage

checkChoice(x, choices, null.ok = FALSE, fmatch = FALSE)

check_choice(x, choices, null.ok = FALSE, fmatch = FALSE)

assertChoice(
  x,
  choices,
  null.ok = FALSE,
  fmatch = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_choice(
  x,
  choices,
  null.ok = FALSE,
  fmatch = FALSE,
  .var.name = vname(x),
  add = NULL
)

testChoice(x, choices, null.ok = FALSE, fmatch = FALSE)

test_choice(x, choices, null.ok = FALSE, fmatch = FALSE)

expect_choice(
  x,
  choices,
  null.ok = FALSE,
  fmatch = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x [any]
Object to check.

choices [atomic]
Set of possible values.

null.ok [logical(1)]
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.

fmatch [logical(1)]
Use the set operations implemented in fmatch in package fastmatch. If fast-
checkClass is not installed, this silently falls back to \texttt{match}. \texttt{fmatch} modifies \( y \) by reference: A hash table is added as attribute which is used in subsequent calls.

\begin{itemize}
  \item \texttt{.var.name} \hspace{1cm} \texttt{character(1)}
  \hspace{1cm} Name of the checked object to print in assertions. Defaults to the heuristic implemented in \texttt{vname}.
  \item \texttt{add} \hspace{1cm} \texttt{[AssertCollection]}
  \hspace{1cm} Collection to store assertion messages. See \texttt{AssertCollection}.
  \item \texttt{info} \hspace{1cm} \texttt{character(1)}
  \hspace{1cm} Extra information to be included in the message for the \texttt{testthat} reporter. See \texttt{expect_that}.
  \item \texttt{label} \hspace{1cm} \texttt{character(1)}
  \hspace{1cm} Name of the checked object to print in messages. Defaults to the heuristic implemented in \texttt{vname}.
\end{itemize}

\textbf{Value}

Depending on the function prefix: If the check is successful, the functions \texttt{assertChoice/assert\_choice} return \( x \) invisibly, whereas \texttt{checkChoice/check\_choice} and \texttt{testChoice/test\_choice} return \texttt{TRUE}. If the check is not successful, \texttt{assertChoice/assert\_choice} throws an error message, \texttt{testChoice/test\_choice} returns \texttt{FALSE}, and \texttt{checkChoice} returns a string with the error message. The function \texttt{expect\_choice} always returns an \texttt{expectation}.

\textbf{Note}

The object \( x \) must be of the same type as the set w.r.t. \texttt{typeof}. Integers and doubles are both treated as numeric.

\textbf{See Also}

Other set: \texttt{checkDisjunct()}, \texttt{checkSetEqual()}, \texttt{checkSubset()}

\textbf{Examples}

\begin{verbatim}
  testChoice("x", letters)
  # x is not converted before the comparison (except for numerics)
  testChoice(factor("a"), "a")
  testChoice(1, "1")
  testChoice(1, as.integer(1))
\end{verbatim}

---

\textbf{checkClass} \hspace{1cm} \textit{Check the class membership of an argument}

\textbf{Description}

Check the class membership of an argument
Usage

checkClass(x, classes, ordered = FALSE, null.ok = FALSE)

check_class(x, classes, ordered = FALSE, null.ok = FALSE)

assertClass(
  x,
  classes,
  ordered = FALSE,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_class(
  x,
  classes,
  ordered = FALSE,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testClass(x, classes, ordered = FALSE, null.ok = FALSE)

test_class(x, classes, ordered = FALSE, null.ok = FALSE)

expect_class(
  x,
  classes,
  ordered = FALSE,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x [any]
Object to check.

classes [character]
Class names to check for inheritance with \texttt{inherits}. \texttt{x} must inherit from all specified classes.

ordered [logical(1)]
Expect \texttt{x} to be specialized in provided order. Default is FALSE.

null.ok [logical(1)]
If set to \texttt{TRUE}, \texttt{x} may also be \texttt{NULL}. In this case only a type check of \texttt{x} is performed, all additional checks are disabled.
checkClass

.bar.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertClass/assert_class return x invisibly, whereas checkClass/check_class and testClass/test_class return TRUE. If the check is not successful, assertClass/assert_class throws an error message, testClass/test_class returns FALSE, and checkClass returns a string with the error message. The function expect_class always returns an expectation.

See Also

Other attributes: checkMultiClass(), checkNamed(), checkNames()

Other classes: checkMultiClass(), checkR6()

Examples

# Create an object with classes "foo" and "bar"
x = 1
class(x) = c("foo", "bar")

# is x of class "foo"?
testClass(x, "foo")

# is x of class "foo" and "bar"?
testClass(x, c("foo", "bar"))

# is x of class "foo" or "bar"?
## Not run:
assert(
  checkClass(x, "foo"),
  checkClass(x, "bar")
)
## End(Not run)

# most specialized as "bar"?
testClass(x, "bar", ordered = TRUE)
checkComplex

Check if an argument is a vector of type complex

Description

Check if an argument is a vector of type complex

Usage

checkComplex(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

check_complex(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

assertComplex(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)
assert_complex(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    names = NULL,
    null.ok = FALSE,
    .var.name = vname(x),
    add = NULL
)

testComplex(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    names = NULL,
    null.ok = FALSE
)

test_complex(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    names = NULL,
    null.ok = FALSE
)

expect_complex(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    names = NULL,
    null.ok = FALSE,
checkComplex

```r
info = NULL,
label = vname(x)
```

**Arguments**

- `x` [any]
  Object to check.
- `any.missing` [logical(1)]
  Are vectors with missing values allowed? Default is TRUE.
- `all.missing` [logical(1)]
  Are vectors with only missing values allowed? Default is TRUE.
- `len` [integer(1)]
  Exact expected length of `x`.
- `min.len` [integer(1)]
  Minimal length of `x`.
- `max.len` [integer(1)]
  Maximal length of `x`.
- `unique` [logical(1)]
  Must all values be unique? Default is FALSE.
- `names` [character(1)]
  Check for names. See `checkNamed` for possible values. Default is “any” which performs no check at all. Note that you can use `checkSubset` to check for a specific set of names.
- `null.ok` [logical(1)]
  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.
- `.var.name` [character(1)]
  Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.
- `add` [AssertCollection]
  Collection to store assertion messages. See `AssertCollection`.
- `info` [character(1)]
  Extra information to be included in the message for the testthat reporter. See `expect_that`.
- `label` [character(1)]
  Name of the checked object to print in messages. Defaults to the heuristic implemented in `vname`.

**Details**

This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_, and NaN.
checkCount

Value

Depending on the function prefix: If the check is successful, the functions assertComplex/assert_complex return `x` invisibly, whereas checkComplex/check_complex and testComplex/test_complex return TRUE. If the check is not successful, assertComplex/assert_complex throws an error message, testComplex/test_complex returns FALSE, and checkComplex returns a string with the error message. The function expect_complex always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

testComplex(1)
testComplex(1+1i)

checkCount

Check if an argument is a count

Description

A count is defined as non-negative integerish value.

Usage

checkCount(
  x,
  na.ok = FALSE,
  positive = FALSE,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE
)

check_count(
  x,
  na.ok = FALSE,
  positive = FALSE,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE
)

assertCount(
  x,
  na.ok = FALSE,
positive = FALSE,
tol = sqrt(.Machine$double.eps),
null.ok = FALSE,
coerce = FALSE,
.var.name = vname(x),
add = NULL
)

assert_count(
  x,
  na.ok = FALSE,
  positive = FALSE,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE,
  coerce = FALSE,
  .var.name = vname(x),
  add = NULL
)

testCount(
  x,
  na.ok = FALSE,
  positive = FALSE,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE
)

test_count(
  x,
  na.ok = FALSE,
  positive = FALSE,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE
)

expect_count(
  x,
  na.ok = FALSE,
  positive = FALSE,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x          [any]
          Object to check.
checkCount

na.ok [logical(1)]
Are missing values allowed? Default is FALSE.

positive [logical(1)]
Must x be positive (>= 1)? Default is FALSE, allowing 0.

tol [double(1)]
Numerical tolerance used to check whether a double or complex can be converted. Default is sqrt(.Machine$double.eps).

null.ok [logical(1)]
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.

coerce [logical(1)]
If TRUE, the input x is returned as integer after an successful assertion.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details

This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value

Depending on the function prefix: If the check is successful, the functions assertCount/assert_count return x invisibly, whereas checkCount/check_count and testCount/test_count return TRUE. If the check is not successful, assertCount/assert_count throws an error message, testCount/test_count returns FALSE, and checkCount returns a string with the error message. The function expect_count always returns an expectation.

Note

To perform an assertion and then convert to integer, use asCount. assertCount will not convert numerics to integer.

See Also

Other scalars: checkFlag(), checkInt(), checkNumber(), checkScalarNA(), checkScalar(), checkString()
Examples

```r
testCount(1)
testCount(-1)
```

---

**checkDataFrame**

*Check if an argument is a data frame*

### Description

Check if an argument is a data frame

### Usage

```r
checkDataFrame(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)
```

```r
check_data_frame(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)
```

```r
assertDataFrame(
  x,
```
checkDataFrame

types = character(0L),
any.missing = TRUE,
all.missing = TRUE,
min.rows = NULL,
max.rows = NULL,
min.cols = NULL,
max.cols = NULL,
nrows = NULL,
ncols = NULL,
row.names = NULL,
col.names = NULL,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

assert_data_frame(
x,
types = character(0L),
any.missing = TRUE,
all.missing = TRUE,
min.rows = NULL,
max.rows = NULL,
min.cols = NULL,
max.cols = NULL,
nrows = NULL,
ncols = NULL,
row.names = NULL,
col.names = NULL,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

testDataFrame(
x,
types = character(0L),
any.missing = TRUE,
all.missing = TRUE,
min.rows = NULL,
max.rows = NULL,
min.cols = NULL,
max.cols = NULL,
nrows = NULL,
ncols = NULL,
row.names = NULL,
col.names = NULL,
null.ok = FALSE
checkDataFrame

)

test_data_frame(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

expect_data_frame(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x [any]
Object to check.

types [character]
Character vector of class names. Each list element must inherit from at least one of the provided types. The types "logical", "integer", "integerish", "double", "numeric", "complex", "character", "factor", "atomic", "vector" "atomicvector", "array", "matrix", "list", "function", "environment" and "null" are supported. For other types inherits is used as a fallback to check x’s inheritance. Defaults to character(0) (no check).

any.missing [logical(1)]
Are missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are columns with only missing values allowed? Default is TRUE.

min.rows [integer(1)]
Minimum number of rows.

max.rows [integer(1)]
Maximum number of rows.

min.cols [integer(1)]
Minimum number of columns.

max.cols [integer(1)]
Maximum number of columns.

nrows [integer(1)]
Exact number of rows.

ncols [integer(1)]
Exact number of columns.

row.names [character(1)]
Check for row names. Default is “NULL” (no check). See checkNamed for possible values. Note that you can use checkSubset to check for a specific set of names.

col.names [character(1)]
Check for column names. Default is “NULL” (no check). See checkNamed for possible values. Note that you can use checkSubset to test for a specific set of names.

null.ok [logical(1)]
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.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertDataFrame/assert_data_frame return x invisibly, whereas checkDataFrame/check_data_frame and testDataFrame/test_data_frame return TRUE. If the check is not successful, assertDataFrame/assert_data_frame throws an error message, testDataFrame/test_data_frame returns FALSE, and checkDataFrame returns a string with the error message. The function expect_data_frame always returns an expectation.
See Also

Other compound: checkArray(), checkDataTable(), checkMatrix(), checkTibble()

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFunction(), checkInteger(), checkIntegerish(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

testDataFrame(iris)
testDataFrame(iris, types = c("numeric", "factor"), min.rows = 1, col.names = "named")

checkDataTable Check if an argument is a data table

Description

Check if an argument is a data table

Usage

checkDataTable(
  x,
  key = NULL,
  index = NULL,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

check_data_table(
  x,
  key = NULL,
  index = NULL,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
checkDataTable

min.rows = NULL,
max.rows = NULL,
min.cols = NULL,
max.cols = NULL,
nrows = NULL,
ncols = NULL,
row.names = NULL,
col.names = NULL,
null.ok = FALSE
)

assertDataTable(
  x,
  key = NULL,
  index = NULL,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_data_table(
  x,
  key = NULL,
  index = NULL,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)
add = NULL
)

testDataTable(
  x,
  key = NULL,
  index = NULL,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

test_data_table(
  x,
  key = NULL,
  index = NULL,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

expect_data_table(
  x,
  key = NULL,
  index = NULL,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
Arguments

x  [any]
    Object to check.
key  [character]
    Expected primary key(s) of the data table.
index  [character]
    Expected secondary key(s) of the data table.
types  [character]
    Character vector of class names. Each list element must inherit from at least one
    of the provided types. The types “logical”, “integer”, “integerish”, “double”,
    “numeric”, “complex”, “character”, “factor”, “atomic”, “vector” “atomicvec-
    tor”, “array”, “matrix”, “list”, “function”, “environment” and “null” are sup-
    ported. For other types inherits is used as a fallback to check x’s inheritance.
    Defaults to character(0) (no check).
any.missing  [logical(1)]
    Are missing values allowed? Default is TRUE.
all.missing  [logical(1)]
    Are matrices with only missing values allowed? Default is TRUE.
min.rows  [integer(1)]
    Minimum number of rows.
max.rows  [integer(1)]
    Maximum number of rows.
min.cols  [integer(1)]
    Minimum number of columns.
max.cols  [integer(1)]
    Maximum number of columns.
nrows  [integer(1)]
    Exact number of rows.
ncols  [integer(1)]
    Exact number of columns.
row.names  [character(1)]
    Check for row names. Default is “NULL” (no check). See checkNamed for possible values. Note that you can use checkSubset to check for a specific set of names.
checkDate

Description

Checks that an object is of class Date.

Value

Depending on the function prefix: If the check is successful, the functions assertDataTable/assert_data_table return x invisibly, whereas checkDataTable/check_data_table and testDataTable/test_data_table return TRUE. If the check is not successful, assertDataTable/assert_data_table throws an error message, testDataTable/test_data_table returns FALSE, and checkDataTable returns a string with the error message. The function expect_data_table always returns an expectation.

See Also

Other compound: checkArray(), checkDataFrame(), checkMatrix(), checkTibble()

Examples

```r
library(data.table)
dt = as.data.table(iris)
setkeyv(dt, "Species")
setkeyv(dt, "Sepal.Length", physical = FALSE)
testDataTable(dt)
testDataTable(dt, key = "Species", index = "Sepal.Length", any.missing = FALSE)
```
Usage

checkDate(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  null.ok = FALSE
)

check_date(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  null.ok = FALSE
)

assertDate(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_date(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

testDate(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  null.ok = FALSE
)

test_date(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  null.ok = FALSE
)

eXpect_date(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  null.ok = FALSE,
  info = NULL,
lable = vname(x)
Arguments

- **x** [any]
  Object to check.
- **lower** [Date]
  All non-missing dates in x must be >= this date. Comparison is done via `Ops.Date`.
- **upper** [Date]
  All non-missing dates in x must be before <= this date. Comparison is done via `Ops.Date`.
- **any.missing** [logical(1)]
  Are vectors with missing values allowed? Default is TRUE.
- **all.missing** [logical(1)]
  Are vectors with only missing values allowed? Default is TRUE.
- **len** [integer(1)]
  Exact expected length of x.
- **min.len** [integer(1)]
  Minimal length of x.
- **max.len** [integer(1)]
  Maximal length of x.
- **unique** [logical(1)]
  Must all values be unique? Default is FALSE.
- **null.ok** [logical(1)]
  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.
- **.var.name** [character(1)]
  Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
- **add** [AssertCollection]
  Collection to store assertion messages. See AssertCollection.
- **info** [character(1)]
  Extra information to be included in the message for the testthat reporter. See expect_that.
- **label** [character(1)]
  Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions `assertAtomic/assert_atomic` return x invisibly, whereas `checkAtomic/check_atomic` and `testAtomic/test_atomic` return TRUE. If the check is not successful, `assertAtomic/assert_atomic` throws an error message, `testAtomic/test_atomic` returns FALSE, and `checkAtomic` returns a string with the error message. The function `expect_atomic` always returns an expectation.
checkDirectoryExists

Check for existence and access rights of directories

Description

Check for existence and access rights of directories

Usage

checkDirectoryExists(x, access = "")
check_directory_exists(x, access = "")
assertDirectoryExists(x, access = "", .var.name = vname(x), add = NULL)
assert_directory_exists(x, access = "", .var.name = vname(x), add = NULL)
testDirectoryExists(x, access = "")
test_directory_exists(x, access = "")
expect_directory_exists(x, access = "", info = NULL, label = vname(x))
checkDirectory(x, access = "")
assertDirectory(x, access = "", .var.name = vname(x), add = NULL)
assert_directory(x, access = "", .var.name = vname(x), add = NULL)
testDirectory(x, access = "")
test_directory(x, access = "")
expect_directory(x, access = "", info = NULL, label = vname(x))

Arguments

x [any]
Object to check.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()
checkDirectoryExists

access [character(1)]
Single string containing possible characters ‘r’, ‘w’ and ‘x’ to force a check for read, write or execute access rights, respectively. Write and executable rights are not checked on Windows.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertDirectoryExists/assert_directory_exists return x invisibly, whereas checkDirectoryExists/check_directory_exists and testDirectoryExists/test_directory_exists return TRUE. If the check is not successful, assertDirectoryExists/assert_directory_exists throws an error message, testDirectoryExists/test_directory_exists returns FALSE, and checkDirectoryExists returns a string with the error message. The function expect_directory_exists always returns an expectation.

Note

The functions without the suffix “exists” are deprecated and will be removed from the package in a future version due to name clashes.

See Also

Other filesystem: checkAccess(), checkFileExists(), checkPathForOutput()

Examples

# Is R's home directory readable?
testDirectory(R.home(), "r")

# Is R's home directory readable and writable?
testDirectory(R.home(), "rw"
CheckDisjunct

Check if an argument is disjunct from a given set

Description

Check if an argument is disjunct from a given set

Usage

checkDisjunct(x, y, fmatch = FALSE)
check_disjunct(x, y, fmatch = FALSE)
assertDisjunct(x, y, fmatch = FALSE, .var.name = vname(x), add = NULL)
assert_disjunct(x, y, fmatch = FALSE, .var.name = vname(x), add = NULL)
testDisjunct(x, y, fmatch = FALSE)
test_disjunct(x, y, fmatch = FALSE)
expect_disjunct(x, y, fmatch = FALSE, info = NULL, label = vname(x))

Arguments

- `x` [any] Object to check.
- `y` [atomic] Other Set.
- `fmatch` [logical(1)] Use the set operations implemented in `fmatch` in package `fastmatch`. If `fastmatch` is not installed, this silently falls back to `match`. `fmatch` modifies `y` by reference: A hash table is added as attribute which is used in subsequent calls.
- `.var.name` [character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.
- `add` [AssertCollection] Collection to store assertion messages. See `AssertCollection`.
- `info` [character(1)] Extra information to be included in the message for the testthat reporter. See `expect_that`.
- `label` [character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in `vname`.
checkDouble

Value

Depending on the function prefix: If the check is successful, the functions assertDisjunct/assert_disjunct return x invisibly, whereas checkDisjunct/check_disjunct and testDisjunct/test_disjunct return TRUE. If the check is not successful, assertDisjunct/assert_disjunct throws an error message, testDisjunct/test_disjunct returns FALSE, and checkDisjunct returns a string with the error message. The function expect_disjunct always returns an expectation.

Note

The object x must be of the same type as the set w.r.t. typeof. Integers and doubles are both treated as numeric.

See Also

Other set: checkChoice(), checkSetEqual(), checkSubset()

Examples

testDisjunct(1L, letters)
testDisjunct(c("a", "z"), letters)

# x is not converted before the comparison (except for numerics)
testDisjunct(factor("a"), "a")
testDisjunct(1, "1")
testDisjunct(1, as.integer(1))

checkDouble

Check that an argument is a vector of type double

Description

Check that an argument is a vector of type double

Usage

checkDouble(
  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,
null.ok = FALSE
)

check_double(
  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,
  null.ok = FALSE
)

assertDouble(
  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,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_double(
  x,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
checkDouble

sorted = FALSE,
names = NULL,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

testDouble(
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,
null.ok = FALSE
)

test_double(
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,
null.ok = FALSE
)

expect_double(
x,
lower = -Inf,
upper = Inf,
finite = FALSE,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
`checkDouble`

```r
max.len = NULL,
unique = FALSE,
sorted = FALSE,
names = NULL,
null.ok = FALSE,
info = NULL,
label = vname(x)
```

**Arguments**

- `x` [any] Object to check.
- `lower` [numeric(1)] Lower value all elements of `x` must be greater than or equal to.
- `upper` [numeric(1)] Upper value all elements of `x` must be lower than or equal to.
- `finite` [logical(1)] Check for only finite values? Default is `FALSE`.
- `any.missing` [logical(1)] Are vectors with missing values allowed? Default is `TRUE`.
- `all.missing` [logical(1)] Are vectors with only missing values allowed? Default is `TRUE`.
- `len` [integer(1)] Exact expected length of `x`.
- `min.len` [integer(1)] Minimal length of `x`.
- `max.len` [integer(1)] Maximal length of `x`.
- `unique` [logical(1)] Must all values be unique? Default is `FALSE`.
- `sorted` [logical(1)] Elements must be sorted in ascending order. Missing values are ignored.
- `names` [character(1)] Check for names. See `checkNamed` for possible values. Default is “any” which performs no check at all. Note that you can use `checkSubset` to check for a specific set of names.
- `null.ok` [logical(1)] 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.
- `var.name` [character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.
- `add` [AssertCollection] Collection to store assertion messages. See `AssertCollection`.  

---

```
Arguments

x [any] Object to check.
lower [numeric(1)] Lower value all elements of x must be greater than or equal to.
upper [numeric(1)] Upper value all elements of x must be lower than or equal to.
finite [logical(1)] Check for only finite values? Default is FALSE.
any.missing [logical(1)] Are vectors with missing values allowed? Default is TRUE.
all.missing [logical(1)] Are vectors with only missing values allowed? Default is TRUE.
len [integer(1)] Exact expected length of x.
min.len [integer(1)] Minimal length of x.
max.len [integer(1)] Maximal length of x.
unique [logical(1)] Must all values be unique? Default is FALSE.
sorted [logical(1)] Elements must be sorted in ascending order. Missing values are ignored.
```
checkEnvironment

info [character(1)]
Extra information to be included in the message for the testthat reporter. See
expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic im-
plemented in vname.

Details

This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value

Depending on the function prefix: If the check is successful, the functions assertDouble/assert_double return x invisibly, whereas checkDouble/check_double and testDouble/test_double return TRUE. If the check is not successful, assertDouble/assert_double throws an error message, testDouble/test_double returns FALSE, and checkDouble returns a string with the error message. The function expect_double always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

testDouble(1)
testDouble(1L)
testDouble(1, min.len = 1, lower = 0)
contains = character(0L),
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

assert_environment(  
  x,
  contains = character(0L),
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testEnvironment(x, contains = character(0L), null.ok = FALSE)

test_environment(x, contains = character(0L), null.ok = FALSE)

expect_environment(  
  x,
  contains = character(0L),
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

**Arguments**

x [any]  
Object to check.

contains [character]  
Vector of object names expected in the environment. Defaults to character(0).

null.ok [logical(1)]  
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.

.var.name [character(1)]  
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]  
Collection to store assertion messages. See AssertCollection.

info [character(1)]  
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]  
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.
checkFactor

Value

Depending on the function prefix: If the check is successful, the functions assertEnvironment/assert_environment return x invisibly, whereas checkEnvironment/check_environment and testEnvironment/test_environment return TRUE. If the check is not successful, assertEnvironment/assert_environment throws an error message, testEnvironment/test_environment returns FALSE, and checkEnvironment returns a string with the error message. The function expect_environment always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

    ee = as.environment(list(a = 1))
    testEnvironment(ee)
    testEnvironment(ee, contains = "a")

checkFactor

Check if an argument is a factor

Description

Check if an argument is a factor

Usage

    checkFactor(
        x,
        levels = NULL,
        ordered = NA,
        empty.levels.ok = TRUE,
        any.missing = TRUE,
        all.missing = TRUE,
        len = NULL,
        min.len = NULL,
        max.len = NULL,
        n.levels = NULL,
        min.levels = NULL,
        max.levels = NULL,
        unique = FALSE,
        names = NULL,
        null.ok = FALSE
    )
check_factor(
  x,
  levels = NULL,
  ordered = NA,
  empty.levels.ok = TRUE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  n.levels = NULL,
  min.levels = NULL,
  max.levels = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

assertFactor(
  x,
  levels = NULL,
  ordered = NA,
  empty.levels.ok = TRUE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  n.levels = NULL,
  min.levels = NULL,
  max.levels = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_factor(
  x,
  levels = NULL,
  ordered = NA,
  empty.levels.ok = TRUE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
checkFactor

max.len = NULL,
n.levels = NULL,
min.levels = NULL,
max.levels = NULL,
unique = FALSE,
names = NULL,
null.ok = FALSE,
.var.name = vname(x),
add = NULL

)testFactor(
  x,
  levels = NULL,
  ordered = NA,
  empty.levels.ok = TRUE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  n.levels = NULL,
  min.levels = NULL,
  max.levels = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

test_factor(
  x,
  levels = NULL,
  ordered = NA,
  empty.levels.ok = TRUE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  n.levels = NULL,
  min.levels = NULL,
  max.levels = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

expect_factor(}
x,
levels = NULL,
ordered = NA,
empty.levels.ok = TRUE,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
n.levels = NULL,
min.levels = NULL,
max.levels = NULL,
unique = FALSE,
names = NULL,
null.ok = FALSE,
info = NULL,
label = vname(x)
)

Arguments

x [any]
Object to check.

levels [character]
Vector of allowed factor levels.

ordered [logical(1)]
Check for an ordered factor? If FALSE or TRUE, checks explicitly for an unordered or ordered factor, respectively. Default is NA which does not perform any additional check.

empty.levels.ok [logical(1)]
Are empty levels allowed? Default is TRUE.

any.missing [logical(1)]
Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

n.levels [integer(1)]
Exact number of factor levels. Default is NULL (no check).

min.levels [integer(1)]
Minimum number of factor levels. Default is NULL (no check).
checkFactor

max.levels [integer(1)]
Maximum number of factor levels. Default is NULL (no check).

unique [logical(1)]
Must all values be unique? Default is FALSE.

names [character(1)]
Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.

null.ok [logical(1)]
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.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertFactor/assert_factor return x invisibly, whereas checkFactor/check_factor and testFactor/test_factor return TRUE. If the check is not successful, assertFactor/assert_factor throws an error message, testFactor/test_factor returns FALSE, and checkFactor returns a string with the error message. The function expect_factor always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

x = factor("a", levels = c("a", "b"))
testFactor(x)
testFactor(x, empty.levels.ok = FALSE)
checkFALSE  
*Check if an argument is FALSE*

**Description**

Simply checks if an argument is FALSE.

**Usage**

```r
checkFALSE(x, na.ok = FALSE)
check_false(x, na.ok = FALSE)
assertFALSE(x, na.ok = FALSE, .var.name = vname(x), add = NULL)
assert_false(x, na.ok = FALSE, .var.name = vname(x), add = NULL)
testFALSE(x, na.ok = FALSE)
test_false(x, na.ok = FALSE)
```

**Arguments**

- `x`  
  [any]
  Object to check.
- `na.ok`  
  [logical(1)]
  Are missing values allowed? Default is FALSE.
- `.var.name`  
  [character(1)]
  Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.
- `add`  
  [AssertCollection]
  Collection to store assertion messages. See `AssertCollection`.

**Value**

Depending on the function prefix: If the check is successful, the functions `assertFALSE`./`assert_false`. return `x` invisibly, whereas `checkFALSE`./`check_false` and `testFALSE`./`test_false`. return TRUE. If the check is not successful, `assertFALSE`./`assert_false`. throws an error message, `testFALSE`./`test_false`. returns FALSE, and `checkFALSE` returns a string with the error message. The function `expect_false`. always returns an `expectation`.

**Examples**

```r
testFALSE(FALSE)
testFALSE(TRUE)
```
checkFileExists

Check existence and access rights of files

Description

Check existence and access rights of files

Usage

```r
checkFileExists(x, access = "", extension = NULL)

check_file_exists(x, access = "", extension = NULL)

assertFileExists(x, access = "", extension = NULL, .var.name = vname(x), add = NULL)

assert_file_exists(x, access = "", extension = NULL, .var.name = vname(x), add = NULL)

testFileExists(x, access = "", extension = NULL)

test_file_exists(x, access = "", extension = NULL)

expect_file_exists(x, access = "", extension = NULL, info = NULL, label = vname(x))

checkFile(x, access = "", extension = NULL)

assertFile(x, access = "", extension = NULL, .var.name = vname(x), add = NULL)

assert_file(x, access = "", extension = NULL, .var.name = vname(x), add = NULL)
```
checkFileExists

testFile(x, access = "", extension = NULL)

expect_file(x, access = "", extension = NULL, info = NULL, label = vname(x))

Arguments

x [any]
Object to check.

access [character(1)]
Single string containing possible characters ‘r’, ‘w’ and ‘x’ to force a check for read, write or execute access rights, respectively. Write and executable rights are not checked on Windows.

extension [character]
Vector of allowed file extensions, matched case insensitive.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertFileExists/assert_file_exists return x invisibly, whereas checkFileExists/check_file_exists and testFileExists/test_file_exists return TRUE. If the check is not successful, assertFileExists/assert_file_exists throws an error message, testFileExists/test_file_exists returns FALSE, and checkFileExists returns a string with the error message. The function expect_file_exists always returns an expectation.

Note

The functions without the suffix “exists” are deprecated and will be removed from the package in a future version due to name clashes. test_file has been unexported already.

See Also

Other filesystem: checkAccess(), checkDirectoryExists(), checkPathForOutput()

Examples

# Check if R's COPYING file is readable
testFileExists(file.path(R.home(), "COPYING"), access = "r")
# Check if R's COPYING file is readable and writable
testFileExists(file.path(R.home(), "COPYING"), access = "rw")

---

**checkFlag**  
*Check if an argument is a flag*

## Description

A flag is defined as single logical value.

## Usage

```r
checkFlag(x, na.ok = FALSE, null.ok = FALSE)
check_flag(x, na.ok = FALSE, null.ok = FALSE)
assertFlag(x, na.ok = FALSE, null.ok = FALSE, .var.name = vname(x), add = NULL)
assert_flag(  
x,  
na.ok = FALSE,  
null.ok = FALSE,  
.var.name = vname(x),  
add = NULL
)
testFlag(x, na.ok = FALSE, null.ok = FALSE)
test_flag(x, na.ok = FALSE, null.ok = FALSE)
epect_flag(x, na.ok = FALSE, null.ok = FALSE, info = NULL, label = vname(x))
```

## Arguments

- **x**  
  [any]
  Object to check.

- **na.ok**  
  [logical(1)]
  Are missing values allowed? Default is FALSE.

- **null.ok**  
  [logical(1)]
  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.

- **.var.name**  
  [character(1)]
  Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.

- **add**  
  [AssertCollection]
  Collection to store assertion messages. See `AssertCollection`.
checkFormula

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details
This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value
Depending on the function prefix: If the check is successful, the functions assertFlag/assert_flag return x invisibly, whereas checkFlag/check_flag and testFlag/test_flag return TRUE. If the check is not successful, assertFlag/assert_flag throws an error message, testFlag/test_flag returns FALSE, and checkFlag returns a string with the error message. The function expect_flag always returns an expectation.

See Also
Other scalars: checkCount(), checkInt(), checkNumber(), checkScalarNA(), checkScalar(), checkString()

Examples

```r
testFlag(TRUE)
testFlag(1)
```

---

checkFormula | Check if an argument is a formula

Description
Check if an argument is a formula

Usage

```r
checkFormula(x, null.ok = FALSE)
check_formula(x, null.ok = FALSE)
assertFormula(x, null.ok = FALSE, .var.name = vname(x), add = NULL)
assert_formula(x, null.ok = FALSE, .var.name = vname(x), add = NULL)
testFormula(x, null.ok = FALSE)
```
test_formula(x, null.ok = FALSE)

expect_formula(x, null.ok = FALSE, info = NULL, label = vname(x))

Arguments

x [any]
Object to check.

null.ok [logical(1)]
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.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertFormula/assert_formula return x invisibly, whereas checkFormula/check_formula and testFormula/test_formula return TRUE. If the check is not successful, assertFormula/assert_formula throws an error message, testFormula/test_formula returns FALSE, and checkFormula returns a string with the error message. The function expect_formula always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

f = Species ~ Sepal.Length + Sepal.Width
checkFormula(f)
checkFunction  

Check if an argument is a function

Description

Check if an argument is a function

Usage

checkFunction(x, args = NULL, ordered = FALSE, nargs = NULL, null.ok = FALSE)

check_function(x, args = NULL, ordered = FALSE, nargs = NULL, null.ok = FALSE)

assertFunction(
  x,
  args = NULL,
  ordered = FALSE,
  nargs = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_function(
  x,
  args = NULL,
  ordered = FALSE,
  nargs = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testFunction(x, args = NULL, ordered = FALSE, nargs = NULL, null.ok = FALSE)

test_function(x, args = NULL, ordered = FALSE, nargs = NULL, null.ok = FALSE)

expect_function(
  x,
  args = NULL,
  ordered = FALSE,
  nargs = NULL,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)
checkFunction

Arguments

- **x** [any]
  Object to check.

- **args**[character]
  Expected formal arguments. Checks that a function has no arguments if set to character(0). Default is NULL (no check).

- **ordered**[logical(1)]
  Flag whether the arguments provided in args must be the first length(args) arguments of the function in the specified order. Default is FALSE.

- **nargs**[integer(1)]
  Required number of arguments, without .... Default is NULL (no check).

- **null.ok**[logical(1)]
  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.

- **.var.name**[character(1)]
  Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

- **add**[AssertCollection]
  Collection to store assertion messages. See AssertCollection.

- **info**[character(1)]
  Extra information to be included in the message for the testthat reporter. See expect_that.

- **label**[character(1)]
  Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertFunction/assert_function return x invisibly, whereas checkFunction/check_function and testFunction/test_function return TRUE. If the check is not successful, assertFunction/assert_function throws an error message, testFunction/test_function returns FALSE, and checkFunction returns a string with the error message. The function expect_function always returns an expectation.

See Also

Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

testFunction(mean)
testFunction(mean, args = "x")
checkInt  

Check if an argument is a single integerish value

Description

Check if an argument is a single integerish value

Usage

```r
checkInt(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE
)
```

```r
check_int(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE
)
```

```r
assertInt(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE,
  coerce = FALSE,
  .var.name = vname(x),
  add = NULL
)
```

```r
assert_int(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE,
  coerce = FALSE,
```
checkInt

```
.var.name = vname(x),
add = NULL
}

testInt(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE
)

test_int(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE
)

expect_int(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  tol = sqrt(.Machine$double.eps),
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)
```

Arguments

- **x** [any]
  Object to check.
- **na.ok** [logical(1)]
  Are missing values allowed? Default is FALSE.
- **lower** [numeric(1)]
  Lower value all elements of x must be greater than or equal to.
- **upper** [numeric(1)]
  Upper value all elements of x must be lower than or equal to.
- **tol** [double(1)]
  Numerical tolerance used to check whether a double or complex can be converted. Default is sqrt(.Machine$double.eps).
- **null.ok** [logical(1)]
checkInt

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.

coerce [logical(1)]
If TRUE, the input x is returned as integer after an successful assertion.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details

This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value

Depending on the function prefix: If the check is successful, the functions assertInt/assert_int return x invisibly, whereas checkInt/check_int and testInt/test_int return TRUE. If the check is not successful, assertInt/assert_int throws an error message, testInt/test_int returns FALSE, and checkInt returns a string with the error message. The function expect_int always returns an expectation.

Note

To perform an assertion and then convert to integer, use asInt. assertInt will not convert numerics to integer.

See Also

Other scalars: checkCount(), checkFlag(), checkNumber(), checkScalarNA(), checkScalar(), checkString()

Examples

testInt(1)
testInt(-1, lower = 0)
checkInteger  

Check if argument is vector of type integer

Description
Check if an argument is vector of type integer

Usage

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

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

```
assertInteger(
  x,
  lower = -Inf,
  upper = Inf,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  null.ok = FALSE
)
```
assert_integer(
    x,
    lower = -Inf,
    upper = Inf,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    sorted = FALSE,
    names = NULL,
    null.ok = FALSE,
    .var.name = vname(x),
    add = NULL
)

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

test_integer(
    x,
    lower = -Inf,
    upper = Inf,
    any.missing = TRUE,
    all.missing = TRUE,
    len = NULL,
checkInteger

min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
names = NULL,
null.ok = FALSE
)

expect_integer(
  x,
  lower = -Inf,
  upper = Inf,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x [any]
Object to check.

lower [numeric(1)]
Lower value all elements of x must be greater than or equal to.

upper [numeric(1)]
Upper value all elements of x must be lower than or equal to.

any.missing [logical(1)]
Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

unique [logical(1)]
Must all values be unique? Default is FALSE.

sorted [logical(1)]
Elements must be sorted in ascending order. Missing values are ignored.
checkInteger

names [character(1)]
Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.

null.ok [logical(1)]
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.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details
This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value
Depending on the function prefix: If the check is successful, the functions assertInteger/assert_integer return x invisibly, whereas checkInteger/check_integer and testInteger/test_integer return TRUE. If the check is not successful, assertInteger/assert_integer throws an error message, testInteger/test_integer returns FALSE, and checkInteger returns a string with the error message. The function expect_integer always returns an expectation.

See Also
asInteger
Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples
testInteger(1L)
testInteger(1.)
testInteger(1:2, lower = 1, upper = 2, any.missing = FALSE)
checkIntegerish  

Check if an object is an integerish vector

Description

An integerish value is defined as value safely convertible to integer. This includes integers and numeric values which are close to an integer w.r.t. a numeric tolerance.

Usage

```r
checkIntegerish(
  x,
  tol = sqrt(.Machine$double.eps),
  lower = -Inf,
  upper = Inf,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  null.ok = FALSE
)
```

```r
check_integerish(
  x,
  tol = sqrt(.Machine$double.eps),
  lower = -Inf,
  upper = Inf,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  null.ok = FALSE
)
```

```r
assertIntegerish(
  x,
  tol = sqrt(.Machine$double.eps),
  lower = -Inf,
  upper = Inf,
```
checkIntegerish

any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
names = NULL,
null.ok = FALSE,
coerce = FALSE,
.var.name = vname(x),
add = NULL
)

assert_integerish(
  x,
tol = sqrt(.Machine$double.eps),
lower = -Inf,
upper = Inf,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
names = NULL,
null.ok = FALSE,
coerce = FALSE,
.var.name = vname(x),
add = NULL
)

testIntegerish(
  x,
tol = sqrt(.Machine$double.eps),
lower = -Inf,
upper = Inf,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
names = NULL,
null.ok = FALSE
)
checkIntegerish

test_integerish(
  x,
  tol = sqrt(.Machine$double.eps),
  lower = -Inf,
  upper = Inf,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  null.ok = FALSE
)

expect_integerish(
  x,
  tol = sqrt(.Machine$double.eps),
  lower = -Inf,
  upper = Inf,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x     [any]
  Object to check.

tol  [double(1)]
  Numerical tolerance used to check whether a double or complex can be converted. Default is \( \sqrt{\text{.Machine}$\text{double.e}} \).

lower [numeric(1)]
  Lower value all elements of \( x \) must be greater than or equal to.

upper [numeric(1)]
  Upper value all elements of \( x \) must be lower than or equal to.

any.missing [logical(1)]
  Are vectors with missing values allowed? Default is \( \text{TRUE} \).
all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

unique [logical(1)]
Must all values be unique? Default is FALSE.

sorted [logical(1)]
Elements must be sorted in ascending order. Missing values are ignored.

names [character(1)]
Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.

null.ok [logical(1)]
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.

coerce [logical(1)]
If TRUE, the input x is returned as integer after an successful assertion.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details

This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value

Depending on the function prefix: If the check is successful, the functions assertIntegerish/assert_integerish return x invisibly, whereas checkIntegerish/check_integerish and testIntegerish/test_integerish return TRUE. If the check is not successful, assertIntegerish/assert_integerish throws an error message, testIntegerish/test_integerish returns FALSE, and checkIntegerish returns a string with the error message. The function expect_integerish always returns an expectation.
Note
To convert from integerish to integer, use \texttt{asInteger}.

See Also
Other basetypes: \texttt{checkArray()}, \texttt{checkAtomicVector()}, \texttt{checkAtomic()}, \texttt{checkCharacter()}, \texttt{checkComplex()}, \texttt{checkDataFrame()}, \texttt{checkDate()}, \texttt{checkDouble()}, \texttt{checkEnvironment()}, \texttt{checkFactor()}, \texttt{checkFormula()}, \texttt{checkFunction()}, \texttt{checkInteger()}, \texttt{checkList()}, \texttt{checkLogical()}, \texttt{checkMatrix()}, \texttt{checkNull()}, \texttt{checkNumeric()}, \texttt{checkPOSIXct()}, \texttt{checkRaw()}, \texttt{checkVector()}

Examples
\begin{verbatim}
testIntegerish(1L)
testIntegerish(1.)
testIntegerish(1:2, lower = 1L, upper = 2L, any.missing = FALSE)
\end{verbatim}

\section*{checkList \textit{Check if an argument is a list}}

\subsection*{Description}
Check if an argument is a list

\subsection*{Usage}
\begin{verbatim}
checkList(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)
\end{verbatim}

\begin{verbatim}
check_list(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)
\end{verbatim}
names = NULL,
null.ok = FALSE
)

assertList(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_list(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testList(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

test_list(}
checkList

x,
types = character(0L),
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
names = NULL,
null.ok = FALSE
)

expect_list(
  x,
types = character(0L),
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
names = NULL,
null.ok = FALSE,
info = NULL,
label = vname(x)
)

Arguments

x [any]
  Object to check.

types [character]
  Character vector of class names. Each list element must inherit from at least one of the provided types. The types "logical", "integer", "integerish", "double", "numeric", "complex", "character", "factor", "atomic", "vector" "atomicvector", "array", "matrix", "list", "function", "environment" and "null" are supported. For other types inherits is used as a fallback to check x's inheritance. Defaults to character(0) (no check).

any.missing [logical(1)]
  Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)]
  Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
  Exact expected length of x.

min.len [integer(1)]
  Minimal length of x.

max.len [integer(1)]
  Maximal length of x.
Must all values be unique? Default is FALSE.

Names Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.

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.

.var.name Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add Collection to store assertion messages. See AssertCollection.

info Extra information to be included in the message for the testthat reporter. See expect_that.

label Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value Depending on the function prefix: If the check is successful, the functions assertList/assert_list return x invisibly, whereas checkList/check_list and testList/test_list return TRUE. If the check is not successful, assertList/assert_list throws an error message, testList/test_list returns FALSE, and checkList returns a string with the error message. The function expect_list always returns an expectation.

Note Contrary to R’s is.list, objects of type pairlist are not recognized as list.

Missingness is defined here as elements of the list being NULL, analogously to anyMissing.

The test for uniqueness does differentiate between the different NA types which are built-in in R. This is required to be consistent with unique while checking scalar missing values. Also see the example.

See Also Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()
checkLogical

Examples

```r
testList(list())
testList(as.list(iris), types = c("numeric", "factor"))

# Missingness
testList(list(1, NA), any.missing = FALSE)
testList(list(1, NULL), any.missing = FALSE)

# Uniqueness differentiates between different NA types:
testList(list(NA, NA), unique = TRUE)
testList(list(NA, NA_real_), unique = TRUE)
```

---

**checkLogical**  
Check if an argument is a vector of type logical

**Description**

Check if an argument is a vector of type logical

**Usage**

```r
checkLogical(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)
```

```r
check_logical(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)
```

```r
assertLogical(
  x,
```
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
names = NULL,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

assert_logical(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testLogical(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

test_logical(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)
checkLogical

) expect_logical(
  x,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x [any] Object to check.

any.missing [logical(1)] Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)] Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)] Exact expected length of x.

min.len [integer(1)] Minimal length of x.

max.len [integer(1)] Maximal length of x.

unique [logical(1)] Must all values be unique? Default is FALSE.

names [character(1)] Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.

null.ok [logical(1)] 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.

.var.name [character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection] Collection to store assertion messages. See AssertCollection.
checkMatrix

info [character(1)]
Extra information to be included in the message for the testthat reporter. See
expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic im-
plemented in vname.

Details
This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_,
and NaN.

Value
Depending on the function prefix: If the check is successful, the functions assertLogical/assert_logical
return x invisibly, whereas checkLogical/check_logical and testLogical/test_logical return TRUE. If the
check is not successful, assertLogical/assert_logical throws an error mes-
sage, testLogical/test_logical returns FALSE, and checkLogical returns a string with the error
message. The function expect_logical always returns an expectation.

See Also
Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(),
checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(),
checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkMatrix(),
checkNull(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()

Examples

testLogical(TRUE)
testLogical(TRUE, min.len = 1)

checkMatrix

Check if an argument is a matrix

Description
Check if an argument is a matrix

Usage
checkMatrix(
x,
mode = NULL,
any.missing = TRUE,
all.missing = TRUE,
min.rows = NULL,
max.rows = NULL,
checkMatrix

min.cols = NULL,
max.cols = NULL,
nrows = NULL,
ncols = NULL,
row.names = NULL,
col.names = NULL,
null.ok = FALSE
)

check_matrix(x,
  mode = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

assertMatrix(x,
  mode = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_matrix(x,
  mode = NULL,
  any.missing = TRUE,
  all.missing = TRUE,


```r
checkMatrix

  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testMatrix(
  x,
  mode = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

test_matrix(
  x,
  mode = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

expect_matrix(
  x,
  mode = NULL,
```
checkMatrix

any.missing = TRUE,
all.missing = TRUE,
min.rows = NULL,
max.rows = NULL,
min.cols = NULL,
max.cols = NULL,
nrows = NULL,
ncols = NULL,
row.names = NULL,
col.names = NULL,
null.ok = FALSE,
info = NULL,
label = vname(x)
)

Arguments

x [any]
Object to check.

mode [character(1)]
Storage mode of the array. Arrays can hold vectors, i.e. “logical”, “integer”,
“integerish”, “double”, “numeric”, “complex”, “character” and “list”. You can
also specify “atomic” here to explicitly prohibit lists. Default is NULL (no check).

any.missing [logical(1)]
Are missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are matrices with only missing values allowed? Default is TRUE.

min.rows [integer(1)]
Minimum number of rows.

max.rows [integer(1)]
Maximum number of rows.

min.cols [integer(1)]
Minimum number of columns.

max.cols [integer(1)]
Maximum number of columns.

nrows [integer(1)]
Exact number of rows.

ncols [integer(1)]
Exact number of columns.

row.names [character(1)]
Check for row names. Default is “NULL” (no check). See checkNamed for
possible values. Note that you can use checkSubset to check for a specific set
of names.

col.names [character(1)]
Check for column names. Default is “NULL” (no check). See checkNamed for
checkMultiClass

Check the class membership of an argument

Possible values. Note that you can use `checkSubset` to test for a specific set of names.

- `null.ok` [logical(1)]
  - 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.

- `.var.name` [character(1)]
  - Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.

- `add` [AssertCollection]
  - Collection to store assertion messages. See `AssertCollection`.

- `info` [character(1)]
  - Extra information to be included in the message for the testthat reporter. See `expect_that`.

- `label` [character(1)]
  - Name of the checked object to print in messages. Defaults to the heuristic implemented in `vname`.

**Value**

Depending on the function prefix: If the check is successful, the functions `assertMatrix/assert_matrix` return `x` invisibly, whereas `checkMatrix/check_matrix` and `testMatrix/test_matrix` return TRUE. If the check is not successful, `assertMatrix/assert_matrix` throws an error message, `testMatrix/test_matrix` returns FALSE, and `checkMatrix` returns a string with the error message. The function `expect_matrix` always returns an expectation.

**See Also**

Other basetypes: `checkArray()`, `checkAtomicVector()`, `checkAtomic()`, `checkCharacter()`, `checkComplex()`, `checkDataFrame()`, `checkDate()`, `checkDouble()`, `checkEnvironment()`, `checkFactor()`, `checkFormula()`, `checkFunction()`, `checkIntegerish()`, `checkInteger()`, `checkList()`, `checkLogical()`, `checkNull()`, `checkNumeric()`, `checkPOSIXct()`, `checkRaw()`, `checkVector()`

Other compound: `checkArray()`, `checkDataFrame()`, `checkDataTable()`, `checkTibble()`

**Examples**

```r
x = matrix(1:9, 3)
colnames(x) = letters[1:3]
testMatrix(x, nrows = 3, min.cols = 1, col.names = "named")
```

```r
checkMultiClass
Check the class membership of an argument
```
Usage

`checkMultiClass(x, classes, null.ok = FALSE)`

`check_multi_class(x, classes, null.ok = FALSE)`

`assertMultiClass(x, classes, null.ok = FALSE, .var.name = vname(x), add = NULL)`

`assert_multi_class(
  x,
  classes,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)`

`testMultiClass(x, classes, null.ok = FALSE)`

`test_multi_class(x, classes, null.ok = FALSE)`

`expect_multi_class(x, classes, null.ok = FALSE, info = NULL, label = vname(x))`

Arguments

- **x** [any]
  
  Object to check.

- **classes** [character]
  
  Class names to check for inheritance with `inherits`. `x` must inherit from any of the specified classes.

- **null.ok** [logical(1)]
  
  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.

- **.var.name** [character(1)]
  
  Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.

- **add** [AssertCollection]
  
  Collection to store assertion messages. See `AssertCollection`.

- **info** [character(1)]
  
  Extra information to be included in the message for the testthat reporter. See `expect_that`.

- **label** [character(1)]
  
  Name of the checked object to print in messages. Defaults to the heuristic implemented in `vname`.

Value

Depending on the function prefix: If the check is successful, the functions `assertMultiClass/assert_multi_class` return `x` invisibly, whereas `checkMultiClass/check_multi_class` and `testMultiClass/test_multi_class`
checkNamed

return TRUE. If the check is not successful, assertMultiClass/assert_multi_class throws an error message, testMultiClass/test_multi_class returns FALSE, and checkMultiClass returns a string with the error message. The function expect_multi_class always returns an expectation.

See Also

Other attributes: checkClass(), checkNamed(), checkNames()
Other classes: checkClass(), checkR6()

Examples

x = 1
class(x) = "bar"
checkMultiClass(x, c("foo", "bar"))
checkMultiClass(x, c("foo", "foobar"))

checkNamed Check if an argument is named

Description

Check if an argument is named

Usage

checkNamed(x, type = "named")
check_named(x, type = "named")
assertNamed(x, type = "named", .var.name = vname(x), add = NULL)
assert_named(x, type = "named", .var.name = vname(x), add = NULL)
testNamed(x, type = "named")
test_named(x, type = "named")

Arguments

x [any]
Object to check.
type [character(1)]
Select the check(s) to perform. “unnamed” checks x to be unnamed. “named” (default) checks x to be named which excludes names to be NA or empty (""). “unique” additionally tests for non-duplicated names. “strict” checks for unique names which comply to R’s variable name restrictions. Note that for zero-length x every name check evaluates to TRUE.
checkNames

Description

Performs various checks on character vectors, usually names.

Usage

```
checkNames(
  x,
  type = "named",
  subset.of = NULL,
  must.include = NULL,
  permutation.of = NULL,
  identical.to = NULL,
  disjunct.from = NULL,
  what = "names"
)
```
checkNames

check_names(
    x,
    type = "named",
    subset.of = NULL,
    must.include = NULL,
    permutation.of = NULL,
    identical.to = NULL,
    disjunct.from = NULL,
    what = "names"
)

assertNames(
    x,
    type = "named",
    subset.of = NULL,
    must.include = NULL,
    permutation.of = NULL,
    identical.to = NULL,
    disjunct.from = NULL,
    what = "names",
    .var.name = vname(x),
    add = NULL
)

assert_names(
    x,
    type = "named",
    subset.of = NULL,
    must.include = NULL,
    permutation.of = NULL,
    identical.to = NULL,
    disjunct.from = NULL,
    what = "names",
    .var.name = vname(x),
    add = NULL
)

testNames(
    x,
    type = "named",
    subset.of = NULL,
    must.include = NULL,
    permutation.of = NULL,
    identical.to = NULL,
    disjunct.from = NULL,
    what = "names"
Arguments

x [character || NULL]
Names to check using rules defined via type.

type [character(1)]
Type of formal check(s) to perform on the names.

unnamed: Checks x to be NULL.
	named: Checks x for regular names which excludes names to be NA or empty (""").

unique: Performs checks like with “named” and additionally tests for non-duplicated names.

strict: Performs checks like with “unique” and additionally fails for names with UTF-8 characters and names which do not comply to R’s variable name restrictions. As regular expression, this is “^[\-.]*[a-zA-Z]+[a-zA-Z0-9_.]*\$”.

type: Same as “strict”, but does not enforce uniqueness.

Note that for zero-length x, all these name checks evaluate to TRUE.

subset.of [character]
Names provided in x must be subset of the set subset.of.

must.include [character]
Names provided in x must be a superset of the set must.include.
permutation.of [character]
Names provided in x must be a permutation of the set permutation.of. Dupli-
cated names in permutation.of are stripped out and duplicated names in x thus lead to a failed check. Use this argument instead of identical.to if the order of the names is not relevant.

identical.to [character]
Names provided in x must be identical to the vector identical.to. Use this argument instead of permutation.of if the order of the names is relevant.

disjunct.from [character]
Names provided in x must may not be present in the vector identical.to.

what [character(1)]
Type of name vector to check, e.g. “names” (default), “colnames” or “rownames”.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertNamed/assert_named return x invisibly, whereas checkNamed/check_named and testNamed/test_named return TRUE. If the check is not successful, assertNamed/assert_named throws an error message, testNamed/test_named returns FALSE, and checkNamed returns a string with the error message. The function expect_named always returns an expectation.

See Also

Other attributes: checkClass(), checkMultiClass(), checkNamed()

Examples

x = 1:3
testNames(x, "unnamed")
names(x) = letters[1:3]
testNames(x, "unique")

assertNames(names(iris), permutation.of = cn)
checkNull  

Check if an argument is NULL

Description
Check if an argument is NULL

Usage
checkNull(x)
check_null(x)
assertNull(x, .var.name = vname(x), add = NULL)
assert_null(x, .var.name = vname(x), add = NULL)
testNull(x)
test_null(x)

Arguments
x                [any]  
Object to check.
.var.name       [character(1)]  
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add             [AssertCollection]  
Collection to store assertion messages. See AssertCollection.

Value
Depending on the function prefix: If the check is successful, the functions assertNull/assert_null return x invisibly, whereas checkNull/check_null and testNull/test_null return TRUE. If the check is not successful, assertNull/assert_null throws an error message, testNull/test_null returns FALSE, and checkNull returns a string with the error message. The function expect_null always returns an expectation.

See Also
Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNumeric(), checkPOSIXct(), checkRaw(), checkVector()
checkNumber

Description

Check if an argument is a single numeric value

Usage

```r
checkNumber(x, na.ok = FALSE, lower = -Inf, upper = Inf, finite = FALSE, null.ok = FALSE)
```

```r
check_number(x, na.ok = FALSE, lower = -Inf, upper = Inf, finite = FALSE, null.ok = FALSE)
```

```r
assertNumber(x, na.ok = FALSE, lower = -Inf, upper = Inf, finite = FALSE, null.ok = FALSE, .var.name = vname(x), add = NULL)
```

```r
assert_number(x, na.ok = FALSE, lower = -Inf, upper = Inf,
```
Arguments

x [any] Object to check.
na.ok [logical(1)] Are missing values allowed? Default is FALSE.
lower [numeric(1)] Lower value all elements of x must be greater than or equal to.
upper [numeric(1)] Upper value all elements of x must be lower than or equal to.
finite [logical(1)] Check for only finite values? Default is FALSE.
null.ok [logical(1)]
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.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details
This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value
Depending on the function prefix: If the check is successful, the functions assertNumber/assert_number return x invisibly, whereas checkNumber/check_number and testNumber/test_number return TRUE. If the check is not successful, assertNumber/assert_number throws an error message, testNumber/test_number returns FALSE, and checkNumber returns a string with the error message. The function expect_number always returns an expectation.

See Also
Other scalars: checkCount(), checkFlag(), checkInt(), checkScalarNA(), checkScalar(), checkString()

Examples

testNumber(1)
testNumber(1:2)

cleanData

checkNumeric | Check 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 check for real integer or double vectors, see checkInteger, checkIntegerish or checkDouble.
Usage

checkNumeric(
  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,
  null.ok = FALSE
)

check_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,
  null.ok = FALSE
)

assertNumeric(
  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,
  null.ok = FALSE,
  .var.name = vname(x),
add = NULL
)

assert_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,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testNumeric(
  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,
  null.ok = FALSE
)

test_numeric(
  x,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
checkNumeric

sorted = FALSE,
names = NULL,
null.ok = FALSE
)

expect_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,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x [any]
Object to check.

lower [numeric(1)]
Lower value all elements of x must be greater than or equal to.

upper [numeric(1)]
Upper value all elements of x must be lower than or equal to.

finite [logical(1)]
Check for only finite values? Default is FALSE.

any.missing [logical(1)]
Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

unique [logical(1)]
Must all values be unique? Default is FALSE.
sorted [logical(1)]
Elements must be sorted in ascending order. Missing values are ignored.

names [character(1)]
Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.

null.ok [logical(1)]
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.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details
This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value
Depending on the function prefix: If the check is successful, the functions assertNumeric/assert_numeric return x invisibly, whereas checkNumeric/check_numeric and testNumeric/test_numeric return TRUE. If the check is not successful, assertNumeric/assert_numeric throws an error message, testNumeric/test_numeric returns FALSE, and checkNumeric returns a string with the error message. The function expect_numeric always returns an expectation.

See Also
Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkPOSIXct(), checkRaw(), checkVector()

Examples

testNumeric(1)
testNumeric(1, min.len = 1, lower = 0)
Description

Check the operating system

Usage

checkOS(os)

check_os(os)

assertOS(os, add = NULL, .var.name = NULL)

assert_os(os, add = NULL, .var.name = NULL)

testOS(os)

test_os(os)

expect_os(os, info = NULL, label = NULL)

Arguments

os [character(1)]
Check the operating system to be in a set with possible elements “windows”, “mac”, “linux” and “solaris”.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertOS/assert_os return x invisibly, whereas checkOS/check_os and testOS/test_os return TRUE. If the check is not successful, assertOS/assert_os throws an error message, testOS/test_os returns FALSE, and checkOS returns a string with the error message. The function expect_os always returns an expectation.
checkPathForOutput

checkPathForOutput

Check if a path is suited for creating an output file

Description

Check if a file path can be used safely to create a file and write to it.
This is checked:

• Does dirname(x) exist?
• Does no file under path x exist?
• Is dirname(x) writable?

Paths are relative to the current working directory.

Usage

checkPathForOutput(x, overwrite = FALSE, extension = NULL)

check_path_for_output(x, overwrite = FALSE, extension = NULL)

assertPathForOutput(
  x,
  overwrite = FALSE,
  extension = NULL,
  .var.name = vname(x),
  add = NULL
)

assert_path_for_output(
  x,
  overwrite = FALSE,
  extension = NULL,
  .var.name = vname(x),
  add = NULL
)

testPathForOutput(x, overwrite = FALSE, extension = NULL)

test_path_for_output(x, overwrite = FALSE, extension = NULL)

expect_path_for_output(
  x,
  overwrite = FALSE,
checkPathForOutput

```r
extension = NULL,
info = NULL,
label = vname(x)
```

**Arguments**

- **x** [any]
  
  Object to check.

- **overwrite** [logical(1)]
  
  If TRUE, an existing file in place is allowed if it it is both readable and writable. Default is FALSE.

- **extension** [character(1)]
  
  Extension of the file, e.g. dQuotetxt or “tar.gz”.

- **.var.name** [character(1)]
  
  Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.

- **add** [AssertCollection]
  
  Collection to store assertion messages. See `AssertCollection`.

- **info** [character(1)]
  
  Extra information to be included in the message for the testthat reporter. See `expect_that`.

- **label** [character(1)]
  
  Name of the checked object to print in messages. Defaults to the heuristic implemented in `vname`.

**Value**

Depending on the function prefix: If the check is successful, the functions `assertPathForOutput/assert_path_for_output` return `x` invisibly, whereas `checkPathForOutput/check_path_for_output` and `testPathForOutput/test_path_for_output` return TRUE. If the check is not successful, `assertPathForOutput/assert_path_for_output` throws an error message, `testPathForOutput/test_path_for_output` returns FALSE, and `checkPathForOutput` returns a string with the error message. The function `expect_path_for_output` always returns an expectation.

**See Also**

Other filesystem: `checkAccess()`, `checkDirectoryExists()`, `checkFileExists()`

**Examples**

```r
# Can we create a file in the tempdir?
testPathForOutput(file.path(tempdir(), "process.log"))
```
checkPOSIXct  

Check that an argument is a date/time object in POSIXct format

Description

Checks that an object is of class **POSIXct**.

Usage

```r
checkPOSIXct(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  null.ok = FALSE
)
```

```r
check_posixct(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  null.ok = FALSE
)
```

```r
assertPOSIXct(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  ```
```r
checkPOSIXct

sorted = FALSE,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

assert_posixct(
x,
lower = NULL,
upper = NULL,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
null.ok = FALSE,
.var.name = vname(x),
add = NULL
)

testPOSIXct(
x,
lower = NULL,
upper = NULL,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
null.ok = FALSE
)

test_posixct(
x,
lower = NULL,
upper = NULL,
any.missing = TRUE,
all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
sorted = FALSE,
null.ok = FALSE
)```
expect_posixct(
  x,
  lower = NULL,
  upper = NULL,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x  [any]
   Object to check.

lower  [Date]
   All non-missing dates in x must be >= this POSIXct time. Must be provided in
   the same timezone as x.

upper  [Date]
   All non-missing dates in x must be <= this POSIXct time. Must be provided in
   the same timezone as x.

any.missing  [logical(1)]
   Are vectors with missing values allowed? Default is TRUE.

all.missing  [logical(1)]
   Are vectors with only missing values allowed? Default is TRUE.

len  [integer(1)]
   Exact expected length of x.

min.len  [integer(1)]
   Minimal length of x.

max.len  [integer(1)]
   Maximal length of x.

unique  [logical(1)]
   Must all values be unique? Default is FALSE.

sorted  [logical(1)]
   Elements must be sorted in ascending order. Missing values are ignored.

null.ok  [logical(1)]
   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.
checkR6

Description

Check if an argument is a R6 class

Usage

```r
checkR6(
  x,
  classes = NULL,
  ordered = FALSE,
  cloneable = NULL,
  public = NULL,
  private = NULL,
  null.ok = FALSE
)
```
checkR6(
    x,
    classes = NULL,
    ordered = FALSE,
    cloneable = NULL,
    public = NULL,
    private = NULL,
    null.ok = FALSE
)

assertR6(
    x,
    classes = NULL,
    ordered = FALSE,
    cloneable = NULL,
    public = NULL,
    private = NULL,
    null.ok = FALSE,
    .var.name = vname(x),
    add = NULL
)

assert_r6(
    x,
    classes = NULL,
    ordered = FALSE,
    cloneable = NULL,
    public = NULL,
    private = NULL,
    null.ok = FALSE,
    .var.name = vname(x),
    add = NULL
)

testR6(
    x,
    classes = NULL,
    ordered = FALSE,
    cloneable = NULL,
    public = NULL,
    private = NULL,
    null.ok = FALSE
)

test_r6(
    x,
    classes = NULL,
    ordered = FALSE,
cloneable = NULL,
public = NULL,
private = NULL,
null.ok = FALSE
)

expect_r6(
  x,
  classes = NULL,
  ordered = FALSE,
  cloneable = NULL,
  public = NULL,
  private = NULL,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

- **x** [any]
  Object to check.

- **classes** [character]
  Class names to check for inheritance with `inherits`. `x` must inherit from all specified classes.

- **ordered** [logical(1)]
  Expect `x` to be specialized in provided order. Default is FALSE.

- **cloneable** [logical(1)]
  If TRUE, check that `x` has a clone method. If FALSE, ensure that `x` is not cloneable.

- **public** [character]
  Names of expected public slots. This includes active bindings.

- **private** [character]
  Names of expected private slots.

- **null.ok** [logical(1)]
  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.

- **.var.name** [character(1)]
  Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.

- **add** [AssertCollection]
  Collection to store assertion messages. See `AssertCollection`.

- **info** [character(1)]
  Extra information to be included in the message for the testthat reporter. See `expect_that`.
checkRaw

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in \texttt{vname}.

\textbf{Value}

Depending on the function prefix: If the check is successful, the functions \texttt{assertClass/assert\_class} return \texttt{x invisibly}, whereas \texttt{checkClass/\_check} and \texttt{testClass/\_test\_class} return \texttt{TRUE}. If the check is not successful, \texttt{assertClass/assert\_class} throws an error message, \texttt{testClass/\_test\_class} returns \texttt{FALSE}, and \texttt{checkClass} returns a string with the error message. The function \texttt{expect\_class} always returns an \texttt{expectation}.

\textbf{See Also}

Other classes: \texttt{checkClass()}, \texttt{checkMultiClass()}

\textbf{Examples}

\begin{verbatim}
library(R6)
generator = R6Class("Bar",
    public = list(a = 5),
    private = list(b = 42),
    active = list(c = function() 99)
)  
x = generator$new()
checkR6(x, "Bar", cloneable = TRUE, public = "a")
\end{verbatim}

---

\textbf{checkRaw} \hspace{1em} \textit{Check if an argument is a raw vector}

\textbf{Description}

Check if an argument is a raw vector

\textbf{Usage}

\begin{verbatim}
checkRaw(
    x,
    len = NULL,
    min.len = NULL,
    max.len = NULL,
    names = NULL,
    null.ok = FALSE
)

check_raw(
    x,
    len = NULL,
\end{verbatim}
```r
checkRaw(
  x,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  names = NULL,
  null.ok = FALSE
)

assertRaw(
  x,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_raw(
  x,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testRaw(
  x,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  names = NULL,
  null.ok = FALSE
)

test_raw(
  x,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  names = NULL,
  null.ok = FALSE
)

expect_raw(
  x,
```
Arguments

- **x** [any]
  - Object to check.
- **len** [integer(1)]
  - Exact expected length of x.
- **min.len** [integer(1)]
  - Minimal length of x.
- **max.len** [integer(1)]
  - Maximal length of x.
- **names** [character(1)]
  - Check for names. See checkNamed for possible values. Default is “any” which performs no check at all. Note that you can use checkSubset to check for a specific set of names.
- **null.ok** [logical(1)]
  - 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.
- **.var.name** [character(1)]
  - Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
- **add** [AssertCollection]
  - Collection to store assertion messages. See AssertCollection.
- **info** [character(1)]
  - Extra information to be included in the message for the testthat reporter. See expect_that.
- **label** [character(1)]
  - Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertRaw/assert_raw return x invisibly, whereas checkRaw/check_raw and testRaw/test_raw return TRUE. If the check is not successful, assertRaw/assert_raw throws an error message, testRaw/test_raw returns FALSE, and checkRaw returns a string with the error message. The function expect_raw always returns an expectation.
checkScalar

See Also
Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkVector()

Examples

testRaw(as.raw(2), min.len = 1L)

checkScalar  Check if an argument is a single atomic value

Description
Check if an argument is a single atomic value

Usage
checkScalar(x, na.ok = FALSE, null.ok = FALSE)
check_scalar(x, na.ok = FALSE, null.ok = FALSE)
assertScalar(
  x,
  na.ok = FALSE,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)
assert_scalar(
  x,
  na.ok = FALSE,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)
testScalar(x, na.ok = FALSE, null.ok = FALSE)
test_scalar(x, na.ok = FALSE, null.ok = FALSE)
expect_scalar(x, na.ok = FALSE, null.ok = FALSE, info = NULL, label = vname(x))
checkScalar

Arguments

x [any] Object to check.
na.ok [logical(1)] Are missing values allowed? Default is FALSE.
null.ok [logical(1)] 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.
.var.name [character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add [AssertCollection] Collection to store assertion messages. See AssertCollection.
info [character(1)] Extra information to be included in the message for the testthat reporter. See expect_that.
label [character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details

This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.

Value

Depending on the function prefix: If the check is successful, the functions assertScalar/assert_scalar return x invisibly, whereas checkScalar/check_scalar and testScalar/test_scalar return TRUE. If the check is not successful, assertScalar/assert_scalar throws an error message, testScalar/test_scalar returns FALSE, and checkScalar returns a string with the error message. The function expect_scalar always returns an expectation.

See Also

Other scalars: checkCount(), checkFlag(), checkInt(), checkNumber(), checkScalarNA(), checkString()

Examples

testScalar(1)
testScalar(1:10)
checkScalarNA

Check if an argument is a single missing value

Description

Check if an argument is a single missing value

Usage

checkScalarNA(x, null.ok = FALSE)
check_scalar_na(x, null.ok = FALSE)
assertScalarNA(x, null.ok = FALSE, .var.name = vname(x), add = NULL)
assert_scalar_na(x, null.ok = FALSE, .var.name = vname(x), add = NULL)
testScalarNA(x, null.ok = FALSE)
test_scalar_na(x, null.ok = FALSE)
expect_scalar_na(x, null.ok = FALSE, info = NULL, label = vname(x))

Arguments

x [any]
Object to check.

null.ok [logical(1)]
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.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.
checkSetEqual

Description

Check if an argument is equal to a given set

Usage

checkSetEqual(x, y, ordered = FALSE, fmatch = FALSE)

check_set_equal(x, y, ordered = FALSE, fmatch = FALSE)

assertSetEqual(
  x,
  y,
  ordered = FALSE,
  fmatch = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_set_equal(
  x,
  y,
  ordered = FALSE,
  fmatch = FALSE,
  .var.name = vname(x),
  add = NULL
)
checkSetEqual

testSetEqual(x, y, ordered = FALSE, fmatch = FALSE)

test_set_equal(x, y, ordered = FALSE, fmatch = FALSE)

expect_set_equal(
  x,
  y,
  ordered = FALSE,
  fmatch = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x [any]
Object to check.

y [atomic]
Set to compare with.

ordered [logical(1)]
Check x to have the same length and order as y, i.e. check using “==” while handling NAs nicely. Default is FALSE.

fmatch [logical(1)]
Use the set operations implemented in fmatch in package fastmatch. If fastmatch is not installed, this silently falls back to match. fmatch modifies y by reference: A hash table is added as attribute which is used in subsequent calls.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertSubset/assert_subset return x invisibly, whereas checkSubset/check_subset and testSubset/test_subset return TRUE. If the check is not successful, assertSubset/assert_subset throws an error message, testSubset/test_subset returns FALSE, and checkSubset returns a string with the error message. The function expect_subset always returns an expectation.
Note
The object x must be of the same type as the set w.r.t. typeof. Integers and doubles are both treated as numeric.

See Also
Other set: checkChoice(), checkDisjunct(), checkSubset()

Examples

testSetEqual(c("a", "b"), c("a", "b"))
testSetEqual(1:3, 1:4)

# x is not converted before the comparison (except for numerics)
testSetEqual(factor("a"), "a")
testSetEqual(1, "1")
testSetEqual(1, as.integer(1))

Description
A string is defined as a scalar character vector.

Usage

checkString(  
  x,  
  na.ok = FALSE,  
  min.chars = NULL,  
  pattern = NULL,  
  fixed = NULL,  
  ignore.case = FALSE,  
  null.ok = FALSE  
)

check_string(  
  x,  
  na.ok = FALSE,  
  min.chars = NULL,  
  pattern = NULL,  
  fixed = NULL,  
  ignore.case = FALSE,  
  null.ok = FALSE  
)
checkString

assertString(
  x,
  na.ok = FALSE,
  min.chars = NULL,
  pattern = NULL,
  fixed = NULL,
  ignore.case = FALSE,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_string(
  x,
  na.ok = FALSE,
  min.chars = NULL,
  pattern = NULL,
  fixed = NULL,
  ignore.case = FALSE,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testString(
  x,
  na.ok = FALSE,
  min.chars = NULL,
  pattern = NULL,
  fixed = NULL,
  ignore.case = FALSE,
  null.ok = FALSE
)

test_string(
  x,
  na.ok = FALSE,
  min.chars = NULL,
  pattern = NULL,
  fixed = NULL,
  ignore.case = FALSE,
  null.ok = FALSE
)

expect_string(
  x,
  na.ok = FALSE,
  min.chars = NULL,
checkString

pattern = NULL,
fixed = NULL,
ignore.case = FALSE,
null.ok = FALSE,
info = NULL,
label = vname(x)
)

Arguments

x [any]
Object to check.

na.ok [logical(1)]
Are missing values allowed? Default is FALSE.

min.chars [integer(1)]
Minimum number of characters for each element of x.

pattern [character(1L)]
Regular expression as used in grepl. All non-missing elements of x must comply to this pattern.

fixed [character(1)]
Substring to detect in x. Will be used as pattern in grepl with option fixed set to TRUE. All non-missing elements of x must contain this substring.

ignore.case [logical(1)]
See grepl. Default is FALSE.

null.ok [logical(1)]
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.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Details

This function does not distinguish between NA, NA_integer_, NA_real_, NA_complex_, NA_character_ and NaN.
checkSubset

Value
Depending on the function prefix: If the check is successful, the functions assertString/assert_string return x invisibly, whereas checkString/check_string and testString/test_string return TRUE. If the check is not successful, assertString/assert_string throws an error message, testString/test_string returns FALSE, and checkString returns a string with the error message. The function expect_string always returns an expectation.

See Also
Other scalars: checkCount(), checkFlag(), checkInt(), checkNumber(), checkScalarNA(), checkScalar()

Examples
testString("a")
testString(letters)

checkSubset Check if an argument is a subset of a given set

Description
Check if an argument is a subset of a given set

Usage
checkSubset(x, choices, empty.ok = TRUE, fmatch = FALSE)
check_subset(x, choices, empty.ok = TRUE, fmatch = FALSE)

assertSubset(
  x,
  choices, 
  empty.ok = TRUE, 
  fmatch = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_subset(
  x,
  choices, 
  empty.ok = TRUE, 
  fmatch = FALSE,
  .var.name = vname(x),
  add = NULL
)
testSubset(x, choices, empty.ok = TRUE, fmatch = FALSE)

test_subset(x, choices, empty.ok = TRUE, fmatch = FALSE)

expect_subset(
  x,
  choices,
  empty.ok = TRUE,
  fmatch = FALSE,
  info = NULL,
  label = vname(x)
)

Arguments

x [any]
Object to check.

choices [atomic]
Set of possible values. May be empty.

empty.ok [logical(1)]
Treat zero-length x as subset of any set choices (this includes NULL)? Default is TRUE.

fmatch [logical(1)]
Use the set operations implemented in fmatch in package fastmatch. If fastmatch is not installed, this silently falls back to match. fmatch modifies y by reference: A hash table is added as attribute which is used in subsequent calls.

.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

info [character(1)]
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value

Depending on the function prefix: If the check is successful, the functions assertSubset/assert_subset return x invisibly, whereas checkSubset/check_subset and testSubset/test_subset return TRUE. If the check is not successful, assertSubset/assert_subset throws an error message, testSubset/test_subset returns FALSE, and checkSubset returns a string with the error message. The function expect_subset always returns an expectation.
Note

The object x must be of the same type as the set w.r.t. typeof. Integers and doubles are both treated as numeric.

See Also

Other set: checkChoice(), checkDisjunct(), checkSetEqual()

Examples

testSubset(c("a", "z"), letters)
testSubset("ab", letters)
testSubset("Species", names(iris))

# x is not converted before the comparison (except for numerics)
testSubset(factor("a"), "a")
testSubset(1, "1")
testSubset(1, as.integer(1))
checkTibble

assertTibble(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_tibble(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)
testTibble(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

test_tibble(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE
)

effect_tibble(
  x,
  types = character(0L),
  any.missing = TRUE,
  all.missing = TRUE,
  min.rows = NULL,
  max.rows = NULL,
  min.cols = NULL,
  max.cols = NULL,
  nrows = NULL,
  ncols = NULL,
  row.names = NULL,
  col.names = NULL,
  null.ok = FALSE,
  info = NULL,


```r
label = vname(x)
```

**Arguments**

- **x** [any]
  - Object to check.

- **types** [character]
  - Character vector of class names. Each list element must inherit from at least one of the provided types. The types “logical”, “integer”, “integerish”, “double”, “numeric”, “complex”, “character”, “factor”, “atomic”, “vector” “atomicvector”, “array”, “matrix”, “list”, “function”, “environment” and “null” are supported. For other types *inherits* is used as a fallback to check x’s inheritance. Defaults to character(0) (no check).

- **any.missing** [logical(1)]
  - Are missing values allowed? Default is TRUE.

- **all.missing** [logical(1)]
  - Are matrices with only missing values allowed? Default is TRUE.

- **min.rows** [integer(1)]
  - Minimum number of rows.

- **max.rows** [integer(1)]
  - Maximum number of rows.

- **min.cols** [integer(1)]
  - Minimum number of columns.

- **max.cols** [integer(1)]
  - Maximum number of columns.

- **nrows** [integer(1)]
  - Exact number of rows.

- **ncols** [integer(1)]
  - Exact number of columns.

- **row.names** [character(1)]
  - Check for row names. Default is “NULL” (no check). See *checkNamed* for possible values. Note that you can use *checkSubset* to check for a specific set of names.

- **col.names** [character(1)]
  - Check for column names. Default is “NULL” (no check). See *checkNamed* for possible values. Note that you can use *checkSubset* to test for a specific set of names.

- **null.ok** [logical(1)]
  - 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.

- **.var.name** [character(1)]
  - Name of the checked object to print in assertions. Defaults to the heuristic implemented in *vname*. 


**checkTRUE**

```r
checkTRUE(x, na.ok = FALSE)
check_true(x, na.ok = FALSE)
assertTRUE(x, na.ok = FALSE, .var.name = vname(x), add = NULL)
assert_true(x, na.ok = FALSE, .var.name = vname(x), add = NULL)
testTRUE(x, na.ok = FALSE)
test_true(x, na.ok = FALSE)
```

**Description**

Simply checks if an argument is TRUE.

**Usage**

- `checkTRUE(x, na.ok = FALSE)`
- `check_true(x, na.ok = FALSE)`
- `assertTRUE(x, na.ok = FALSE, .var.name = vname(x), add = NULL)`
- `assert_true(x, na.ok = FALSE, .var.name = vname(x), add = NULL)`
- `testTRUE(x, na.ok = FALSE)`
- `test_true(x, na.ok = FALSE)`
checkVector

Arguments

- **x**: [any]
  - Object to check.
- **na.ok**: [logical(1)]
  - Are missing values allowed? Default is FALSE.
- **.var.name**: [character(1)]
  - Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.
- **add**: [AssertCollection]
  - Collection to store assertion messages. See `AssertCollection`.

Value

Depending on the function prefix: If the check is successful, the functions `assertTRUE`/`assert_true` return `x` invisibly, whereas `checkTRUE`/`check_true` and `testTRUE`/`test_true` return TRUE. If the check is not successful, `assertTRUE`/`assert_true` throws an error message, `testTRUE`/`test_true` returns FALSE, and `checkTRUE` returns a string with the error message. The function `expect_true` always returns an `expectation`.

Examples

```
testTRUE(TRUE)
testTRUE(FALSE)
```

---

checkVector **Check if an argument is a vector**

Description

Check if an argument is a vector

Usage

```
checkVector(
  x,
  strict = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)
```
checkVector

check_vector(
  x,
  strict = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

assertVector(
  x,
  strict = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

assert_vector(
  x,
  strict = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE,
  .var.name = vname(x),
  add = NULL
)

testVector(
  x,
  strict = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
len = NULL,
min.len = NULL,
max.len = NULL,
unique = FALSE,
names = NULL,
null.ok = FALSE
)

test_vector(
  x,
  strict = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  names = NULL,
  null.ok = FALSE
)

Arguments

x [any]
Object to check.

strict [logical(1)]
May the vector have additional attributes? If TRUE, mimics the behavior of
is.vector. Default is FALSE which allows e.g. factors or data.frames to
be recognized as vectors.

any.missing [logical(1)]
Are vectors with missing values allowed? Default is TRUE.

all.missing [logical(1)]
Are vectors with only missing values allowed? Default is TRUE.

len [integer(1)]
Exact expected length of x.

min.len [integer(1)]
Minimal length of x.

max.len [integer(1)]
Maximal length of x.

unique [logical(1)]
Must all values be unique? Default is FALSE.

names [character(1)]
Check for names. See checkNamed for possible values. Default is “any” which
performs no check at all. Note that you can use checkSubset to check for a
specific set of names.
null.ok [logical(1)]
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.
.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add [AssertCollection]
Collection to store assertion messages. See AssertCollection.

Value
Depending on the function prefix: If the check is successful, the functions assertVector/assert_vector return x invisibly, whereas checkVector/check_vector and testVector/test_vector return TRUE.
If the check is not successful, assertVector/assert_vector throws an error message, testVector/test_vector returns FALSE, and checkVector returns a string with the error message. The function expect_vector always returns an expectation.

See Also
Other basetypes: checkArray(), checkAtomicVector(), checkAtomic(), checkCharacter(), checkComplex(), checkDataFrame(), checkDate(), checkDouble(), checkEnvironment(), checkFactor(), checkFormula(), checkFunction(), checkIntegerish(), checkInteger(), checkList(), checkLogical(), checkMatrix(), checkNull(), checkNumeric(), checkPOSIXct(), checkRaw()
Other atomicvector: checkAtomicVector(), checkAtomic()

Examples

testVector(letters, min.len = 1L, any.missing = FALSE)

Description
makeAssertion is the internal function used to evaluate the result of a check and throw an exception if necessary. makeAssertionFunction can be used to automatically create an assertion function based on a check function (see example).

Usage

makeAssertion(x, res, var.name, collection)

makeAssertionFunction(
  check.fun, 
  c.fun = NULL, 
  use.namespace = TRUE, 
  coerce = FALSE, 
  env = parent.frame() 
)
makeAssertion

Arguments

x [any]
Object to check.

res [TRUE | character(1)]
The result of a check function: TRUE for successful checks, and an error message as string otherwise.

var.name [character(1)]
The custom name for x as passed to any assert* function. Defaults to a heuristic name lookup.

collection [AssertCollection]
If an AssertCollection is provided, the error message is stored in it. If NULL, an exception is raised if res is not TRUE.

check.fun [function]
Function which checks the input. Must return TRUE on success and a string with the error message otherwise.

c.fun [character(1)]
If not NULL, instead of calling the function check.fun, use .Call to call a C function “c.fun” with the identical set of parameters. The C function must be registered as a native symbol, see .Call. Useful if check.fun is just a simple wrapper.

use.namespace [logical(1)]
Call functions of checkmate using its namespace explicitly. Can be set to FALSE so save some microseconds, but the checkmate package needs to be imported. Default is TRUE.

coerce [logical(1)]
If TRUE, injects some lines of code to convert numeric values to integer after an successful assertion. Currently used in assertCount, assertInt and assertIntegerish.

env [environment]
The environment of the created function. Default is the parent.frame.

Value

makeAssertion invisibly returns the checked object if the check was successful, and an exception is raised (or its message stored in the collection) otherwise. makeAssertionFunction returns a function.

See Also

Other CustomConstructors: makeExpectation(), makeTest()

Examples

# Simple custom check function
checkFalse = function(x) if (!identical(x, FALSE)) "Must be FALSE" else TRUE
# Create the respective assert function
assertFalse = function(x, .var.name = vname(x), add = NULL) {
  res = checkFalse(x)
  makeAssertion(x, res, .var.name, add)
}

# Alternative: Automatically create such a function
assertFalse = makeAssertionFunction(checkFalse)
print(assertFalse)

---

## `makeExpectation` 

**Turn a Check into an Expectation**

**Description**

`makeExpectation` is the internal function used to evaluate the result of a check and turn it into an expectation. `makeExceptionFunction` can be used to automatically create an expectation function based on a check function (see example).

**Usage**

```r
makeExpectation(x, res, info, label)
```

```r
def makeExpectationFunction(
  check.fun,
  c.fun = NULL,
  use.namespace = FALSE,
  env = parent.frame()
)
```

**Arguments**

- **x**  
  Object to check.

- **res**  
  `[TRUE | character(1)]`  
  The result of a check function: TRUE for successful checks, and an error message as string otherwise.

- **info**  
  `character(1)`  
  See `expect_that`

- **label**  
  `character(1)`  
  See `expect_that`

- **check.fun**  
  `function`  
  Function which checks the input. Must return TRUE on success and a string with the error message otherwise.
makeTest

## Description

`makeTest` is the internal function used to evaluate the result of a check and throw an exception if necessary. This function is currently only a stub and just calls `isTRUE`. `makeTestFunction` can be used to automatically create an assertion function based on a check function (see example).

## Usage

```r
makeTest(res)
makeTestFunction(check.fun, c.fun = NULL, env = parent.frame())
```

## Usage

```r
c.fun [character(1)]
If not NULL, instead of calling the function check.fun, use .Call to call a C function “c.fun” with the identical set of parameters. The C function must be registered as a native symbol, see .Call. Useful if check.fun is just a simple wrapper.

use.namespace [logical(1)]
Call functions of `checkmate` using its namespace explicitly. Can be set to `FALSE` so save some microseconds, but the checkmate package needs to be imported. Default is `TRUE`.

env [environment]
The environment of the created function. Default is the `parent.frame`.

## Value

`makeExpectation` invisibly returns the checked object. `makeExpectationFunction` returns a function.

## See Also

Other CustomConstructors: `makeAssertion()`, `makeTest()`

## Examples

```r
# Simple custom check function
checkFalse = function(x) if (!identical(x, FALSE)) "Must be FALSE" else TRUE

# Create the respective expect function
expect_false = function(x, info = NULL, label = vname(x)) {
  res = checkFalse(x)
  makeExpectation(x, res, info = info, label = label)
}

# Alternative: Automatically create such a function
expect_false = makeExpectationFunction(checkFalse)
print(expect_false)
```
**Arguments**

- **res**
  
  [TRUE | character(1)]
  
  The result of a check function: TRUE for successful checks, and an error message as string otherwise.

- **check.fun**
  
  [function]
  
  Function which checks the input. Must return TRUE on success and a string with the error message otherwise.

- **c.fun**
  
  [character(1)]
  
  If not NULL, instead of calling the function check.fun, use .Call to call a C function “c.fun” with the identical set of parameters. The C function must be registered as a native symbol, see .Call. Useful if check.fun is just a simple wrapper.

- **env**
  
  [environment]
  
  The environment of the created function. Default is the parent.frame.

**Value**

makeTest returns TRUE if the check is successful and FALSE otherwise. makeTestFunction returns a function.

**See Also**

Other CustomConstructors: makeAssertion(), makeExpectation()

**Examples**

```r
# Simple custom check function
checkFalse = function(x) if (!identical(x, FALSE)) "Must be FALSE" else TRUE

# Create the respective test function
testFalse = function(x) {
  res = checkFalse(x)
  makeTest(res)
}

# Alternative: Automatically create such a function
testFalse = makeTestFunction(checkFalse)
print(testFalse)
```

---

**matchArg**

*Partial Argument Matching*

**Description**

This is an extensions to match.arg with support for AssertCollection. The behavior is very similar to match.arg, except that NULL is not a valid value for x.
Usage

```r
matchArg(x, choices, several.ok = FALSE, .var.name = vname(x), add = NULL)
```

Arguments

- **x** [character]
  User provided argument to match.
- **choices** [character()]
  Candidates to match `x` with.
- **several.ok** [logical(1)]
  If TRUE, multiple matches are allowed, cf. `match.arg`.
- **.var.name** [character(1)]
  Name of the checked object to print in error messages. Defaults to the heuristic implemented in `vname`.
- **add** [AssertCollection]
  Collection to store assertions. See `AssertCollection`.

Value

Subset of `choices`.

Examples

```r
matchArg("k", choices = c("kendall", "pearson"))
```

---

**qassert**

*Quick argument checks on (builtin) R types*

Description

The provided functions parse rules which allow to express some of the most frequent argument checks by typing just a few letters.

Usage

```r
qassert(x, rules, .var.name = vname(x))
qtest(x, rules)
qexpect(x, rules, info = NULL, label = vname(x))
```
Arguments

- **x** [any]
  - Object the check.

- **rules** [character]
  - Set of rules. See details.

- **.var.name** [character(1)]
  - Name of the checked object to print in error messages. Defaults to the heuristic implemented in `vname`.

- **info** [character(1)]
  - Extra information to be included in the message for the testthat reporter. See `expect_that`.

- **label** [character(1)]
  - Name of the checked object to print in messages. Defaults to the heuristic implemented in `vname`.

Details

The rule is specified in up to three parts.

1. Class and missingness check. The first letter is an abbreviation for the class. If it is provided uppercase, missing values are prohibited. Supported abbreviations:

   - `[bB]` Bool / logical.
   - `[iI]` Integer.
   - `[xX]` Integerish (numeric convertible to integer, see `checkIntegerish`).
   - `[rR]` Real / double.
   - `[cC]` Complex.
   - `[nN]` Numeric (integer or double).
   - `[sS]` String / character.
   - `[fF]` Factor
   - `[aA]` Atomic.
   - `[vV]` Atomic vector (see `checkAtomicVector`).
   - `[lL]` List. Missingness is defined as NULL element.
   - `[mM]` Matrix.
   - `[dD]` Data.frame. Missingness is checked recursively on columns.
   - `[pP]` POSIXct date.
   - `[e]` Environment.
   - `[0]` NULL.
   - `[*]` placeholder to allow any type.

   Note that the check for missingness does not distinguish between `NaN` and `NA`. Infinite values are not treated as missing, but can be caught using boundary checks (part 3).

2. Length definition. This can be one of

   - `[*]` any length.
   - `[?]` length of zero or one,
   - `[+]` length of at least one, or
   - `[0-9]+` exact length specified as integer.
Preceding the exact length with one of the comparison operators \(=/=, <, \leq, \geq \) or \(> \) is also supported.

3. Range check as two numbers separated by a comma, enclosed by square brackets (endpoint included) or parentheses (endpoint excluded). For example, \([0, 3)\) results in \(\{x \geq 0 \&\& x < 3\}\). The lower and upper bound may be omitted which is the equivalent of a negative or positive infinite bound, respectively. By definition \([0,]\) contains Inf, while \([0,)\) does not. The same holds for the left (lower) boundary and \(-\text{Inf}\). E.g., the rule “N1()” checks for a single finite numeric which is not NA, while “N1[)” allows \(-\text{Inf}\).

**Value**

\texttt{qassert} throws an \texttt{R} exception if object \texttt{x} does not comply to at least one of the rules and returns the tested object invisibly otherwise. \texttt{qtest} behaves the same way but returns \texttt{FALSE} if none of the rules comply. \texttt{qexpect} is intended to be inside the unit test framework \texttt{testthat} and returns an expectation.

**Note**

The functions are inspired by the blog post of Bogumił Kamiński: \url{http://rsnippets.blogspot.de/2013/06/testing-function-aguments-in-gnu-r.html}. The implementation is mostly written in C to minimize the overhead.

**See Also**

\texttt{qtestr} and \texttt{qassertr} for efficient checks of list elements and data frame columns.

**Examples**

```r
# logical of length 1
qtest(NA, "b1")

# logical of length 1, NA not allowed
qtest(NA, "B1")

# logical of length 0 or 1, NA not allowed
qtest(TRUE, "B?")

# numeric with length > 0
qtest(runif(10), "n+")

# integer with length > 0, NAs not allowed, all integers \(\geq 0\) and < Inf
qtest(1:3, "I+[0,)")

# either an empty list or a character vector with \(\leq 5\) elements
qtest(c("10", "s<=5"))

# data frame with at least one column and no missing value in any column
qtest(iris, "D++")
```

qassertr: Quick recursive arguments checks on lists and data frames

Description
These functions are the tuned counterparts of qtest, qassert and qexpect tailored for recursive checks of list elements or data frame columns.

Usage
qassertr(x, rules, .var.name = vname(x))
qtestr(x, rules, depth = 1L)
qexpectr(x, rules, info = NULL, label = vname(x))

Arguments
x [list or data.frame]  
List or data frame to check for compliance with at least one of rules. See details of qtest for rule explanation.

rules [character]  
Set of rules. See qtest

.var.name [character(1)]  
Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.

depth [integer(1)]  
Maximum recursion depth. Defaults to “1” to directly check list elements or data frame columns. Set to a higher value to check lists of lists of elements.

info [character(1)]  
Extra information to be included in the message for the testthat reporter. See expect_that.

label [character(1)]  
Name of the checked object to print in messages. Defaults to the heuristic implemented in vname.

Value
See qassert.

See Also
qtest, qassert
Examples

# All list elements are integers with length >= 1?
qtestr(as.list(1:10), "i+")

# All list elements (i.e. data frame columns) are numeric?
qtestr(iris, "n")

# All list elements are numeric, w/o NAs?
qtestr(list(a = 1:3, b = rnorm(1), c = letters), "N+")

# All list elements are numeric OR character
qtestr(list(a = 1:3, b = rnorm(1), c = letters), c("N+", "S+"))

register_test_backend

Select Backend for Unit Tests

Description

Allows to explicitly select a backend for the unit tests. Currently supported are “testthat” and “tinytest”. The respective package must be installed and are loaded (but not attached).

If this function is not explicitly called, defaults to “testthat” unless the “tinytest”’s namespace is loaded.

Usage

register_test_backend(name)

Arguments

name [character(1)]
   "testthat" or "tinytest".

Value

NULL (invisibly).

vname

Lookup a variable name

Description

Tries to heuristically determine the variable name of x in the parent frame with a combination of deparse and substitute. Used for checkmate’s error messages.

Usage

vname(x)
Arguments

x [ANY]
Object.

Value

character(1) Variable name.

description

A quick C implementation for “which.first” (head(which(x), 1)) and “which.last” (tail(which(x), 1)).

Usage

wf(x, use.names = TRUE)
wl(x, use.names = TRUE)

Arguments

x [logical]
Logical vector.
use.names [logical(1)]
If TRUE and x is named, the result is also named.

Value

integer(1) | integer(0). Returns the index of the first/last TRUE value in x or an empty integer vector if none is found. NAs are ignored.

Examples

wf(c(FALSE, TRUE))
wlf(c(FALSE, FALSE))
wf(NA)
Coalesce operator

Description

Returns the left hand side if not missing nor NULL, and the right hand side otherwise.

Usage

lhs %??% rhs

Arguments

lhs [any]  
Left hand side of the operator. Is returned if not missing or NULL.

rhs [any]  
Right hand side of the operator. Is returned if lhs is missing or NULL.

Value

Either lhs or rhs.

Examples

print(NULL %??% 1 %??% 2)  
print(names(iris) %??% letters[seq_len(ncol(iris))])
assertComplex (checkComplex), 32
assertCount (checkCount), 8, 140
assertDataFrame (checkDataFrame), 35
assertDataTable (checkDataTable), 42
assertDirectory (checkDirectoryExists), 50
assertDirectoryExists (checkDirectoryExists), 50
assertDisjunct (checkDisjunct), 52
assertDouble (checkDouble), 53
assertEnvironment (checkEnvironment), 57
assertFactor (checkFactor), 59
assertFALSE (checkFALSE), 64
assertFile (checkFileExists), 65
assertFlag (checkFlag), 67
assertFormula (checkFormula), 68
assertFunction (checkFunction), 70
assertInt (checkInt), 8, 140
assertInteger (checkInteger), 72
assertIntegerish (checkIntegerish), 75
assertList (checkList), 83
assertLogical (checkLogical), 87
assertMatrix (checkMatrix), 90
assertMultiClass (checkMultiClass), 94
assertNamed (checkNamed), 96
assertNames (checkNames), 97
assertNull (checkNull), 101
assertNumber (checkNumber), 102
assertNumeric (checkNumeric), 104
assertOS (checkOS), 109
assertPathForOutput (checkPathForOutput), 110
assertPOSIXct (checkPOSIXct), 112
assertR6 (checkR6), 115
assertRaw (checkRaw), 118
assertScalar (checkScalar), 121
assertScalarNA (checkScalarNA), 123
assertSetEqual (checkSetEqual), 124
assertString (checkString), 126
assertSubset (checkSubset), 129
assertTibble (checkTibble), 131
assertTRUE (checkTRUE), 135
assertVector (checkVector), 136
check_access (checkAccess), 13
check_array (checkArray), 14
check_atomic (checkAtomic), 17
check_atomic_vector (checkAtomicVector), 20
check_character (checkCharacter), 23
check_choice (checkChoice), 27
check_class (checkClass), 29
check_complex (checkComplex), 32
check_count (checkCount), 35
check_data_frame (checkDataFrame), 38
check_data_table (checkDataTable), 42
check_date (checkDate), 46
check_directory_exists (checkDirectoryExists), 50
check_disjunct (checkDisjunct), 52
check_double (checkDouble), 53
check_environment (checkEnvironment), 57
check_factor (checkFactor), 59
check_false (checkFALSE), 64
check_file_exists (checkFileExists), 65
check_flag (checkFlag), 67
check_function (checkFunction), 68
check_int (checkInt), 70
check_integer (checkInteger), 72
check_integerish (checkIntegerish), 75
check_list (checkList), 83
check_logical (checkLogical), 87
check_matrix (checkMatrix), 90
check_multi_class (checkMultiClass), 94
check_named (checkNamed), 96
check_names (checkNames), 97
check_null (checkNull), 101
check_number (checkNumber), 102
check_numeric (checkNumeric), 104
check_os (checkOS), 109
check_path_for_output (checkPathForOutput), 110
check_posixct (checkPOSIXct), 112
check_r6 (checkR6), 115
check_raw (checkRaw), 118
check_scalar (checkScalar), 121
check_scalar_na (checkScalarNA), 123
INDEX

checkSubset, 5, 29, 53, 124, 131
checkString, 4, 37, 68, 74, 104, 122, 124, 126
checkSubset, 5, 10, 19, 26, 29, 34, 41, 45, 46, 53, 56, 63, 78, 82, 86, 89, 93, 94, 108, 120, 126, 129, 134, 138
checkTibble, 5, 17, 42, 46, 94, 131
checkTRUE, 135

Date, 46, 49, 114
deparse, 148

expect_access (checkAccess), 13
expect_array (checkArray), 14
expect_atomic (checkAtomic), 17
expect_atomic_vector (checkAtomicVector), 20
expect_character (checkCharacter), 23
expect_choice (checkChoice), 27
expect_class (checkClass), 29
expect_complex (checkComplex), 32
expect_count (checkCount), 35
expect_data_frame (checkDataFrame), 38
expect_data_table (checkDataTable), 42
expect_date (checkDate), 46
expect_directory (checkDirectoryExists), 50
expect_directory_exists (checkDirectoryExists), 50
expect_disjunct (checkDisjunct), 52
expect_double (checkDouble), 53
expect_environment (checkEnvironment), 57
expect_factor (checkFactor), 59
expect_file (checkFileExists), 65
expect_file_exists (checkFileExists), 65
expect_flag (checkFlag), 67
expect_formula (checkFormula), 68
expect_function (checkFunction), 70
expect_int (checkInt), 72
expect_integer (checkInteger), 75
expect_integerish (checkIntegerish), 79
expect_list (checkList), 83
expect_logical (checkLogical), 87
expect_matrix (checkMatrix), 90
expect_multi_class (checkMultiClass), 94
expect_names (checkNames), 97
expect_number (checkNumber), 102
expect_numeric (checkNumeric), 104
expect_os (checkOS), 109
expect_path_for_output (checkPathForOutput), 110
expect_posixct (checkPOSIXct), 112
expect_r6 (checkR6), 115
expect_raw (checkRaw), 118
expect_scalar (checkScalar), 121
expect_scalar_na (checkScalarNA), 123
expect_set_equal (checkSetEqual), 124
expect_string (checkString), 126
expect_subset (checkSubset), 129
expect_that, 13, 16, 19, 22, 27, 29, 31, 34, 37, 41, 46, 49, 51, 52, 57, 58, 63, 66, 68, 69, 71, 74, 78, 82, 86, 90, 94, 95, 100, 104, 108, 109, 111, 115, 117, 120, 122, 123, 125, 128, 130, 135, 141, 145, 147
expect_tibble (checkTibble), 131

fmatch, 28, 29, 52, 125, 130
grepl, 26, 128

inherits, 30, 40, 45, 85, 95, 117, 134
is.atomic, 6–8, 17, 20
is.list, 86
is.numeric, 104
is.vector, 20, 138
isTRUE, 142

makeAssertCollection (AssertCollection), 12
makeAssertion, 139, 142, 143
makeAssertionFunction (makeAssertion), 139
makeExpectation, 140, 141, 143
makeExpectationFunction (makeExpectation), 141
makeTest, 140, 142, 142
makeTestFunction (makeTest), 142
INDEX

match, 29, 52, 125, 130
match.arg, 143, 144
matchArg, 143
Ops.Date, 49
pairlist, 86
parent.frame, 140, 142, 143
POSIXct, 112
qassert, 5, 144, 147
qassertr, 5, 146, 147
qexpect, 147
qexpect (qassert), 144
qexpectr (qassertr), 147
qtest, 147
qtest (qassert), 144
qtestr, 146
qtester (qassertr), 147
register_test_backend, 148
reportAssertions (AssertCollection), 12
substitute, 148
test_access (checkAccess), 13
test_array (checkArray), 14
test_atomic (checkAtomic), 17
test_atomic_vector (checkAtomicVector), 20
test_character (checkCharacter), 23
test_choice (checkChoice), 27
test_class (checkClass), 29
test_complex (checkComplex), 32
test_count (checkCount), 35
test_data_frame (checkDataFrame), 38
test_data_table (checkDataTable), 42
test_date (checkDate), 46
test_directory (checkDirectoryExists), 50
test_directory_exists (checkDirectoryExists), 50
test_disjunct (checkDisjunct), 52
test_double (checkDouble), 53
test_environment (checkEnvironment), 57
test_factor (checkFactor), 59
test_false (checkFALSE), 64
test_file_exists (checkFileExists), 65
test_flag (checkFlag), 67
test_formula (checkFormula), 68
test_function (checkFunction), 70
test_int (checkInt), 72
test_integer (checkInteger), 75
test_integerish (checkIntegerish), 79
test_list (checkList), 83
test_logical (checkLogical), 87
test_matrix (checkMatrix), 90
test_multi_class (checkMultiClass), 94
test_named (checkNamed), 96
test_names (checkNames), 97
test_null (checkNull), 101
test_number (checkNumber), 102
test_numeric (checkNumeric), 104
test_os (checkOS), 109
test_path_for_output (checkPathForOutput), 110
test_posixct (check POSIXct), 112
test_r6 (checkR6), 115
test_raw (checkRaw), 118
test_scalar (checkScalar), 121
test_scalar_na (checkScalarNA), 123
test_set_equal (checkSetEqual), 124
test_string (checkString), 126
test_subset (checkSubset), 129
test_tibble (checkTibble), 131
test_true (checkTRUE), 135
test_vector (checkVector), 136
testAccess (checkAccess), 13
testArray (checkArray), 14
testAtomic (checkAtomic), 17
testAtomicVector (checkAtomicVector), 20
testCharacter (checkCharacter), 23
testChoice (checkChoice), 27
testClass (checkClass), 29
testComplex (checkComplex), 32
testCount (checkCount), 35
testDataFrame (checkDataFrame), 38
testDataTable (checkDataTable), 42
testDate (checkDate), 46
testDirectory (checkDirectoryExists), 50
testDirectoryExists (checkDirectoryExists), 50
testDisjunct (checkDisjunct), 52
testDouble (checkDouble), 53
testEnvironment (checkEnvironment), 57
testFactor (checkFactor), 59
testFALSE (checkFALSE), 64
testFile (checkFileExists), 65
testFileExists (checkFileExists), 65
testFlag (checkFlag), 67
testFormula (checkFormula), 68
testFunction (checkFunction), 70
testInt (checkInt), 72
testInteger (checkInteger), 75
testIntegerish (checkIntegerish), 79
testList (checkList), 83
testLogical (checkLogical), 87
testMatrix (checkMatrix), 90
testMultiClass (checkMultiClass), 94
testNamed (checkNamed), 96
testNames (checkNames), 97
testNull (checkNull), 101
testNumber (checkNumber), 102
testNumeric (checkNumeric), 104
testOS (checkOS), 109
testPathForOutput (checkPathForOutput), 110
testPOSIXct (checkPOSIXct), 112
testR6 (checkR6), 115
testRaw (checkRaw), 118
testScalar (checkScalar), 121
testScalarNA (checkScalarNA), 123
testSetEqual (checkSetEqual), 124
testString (checkString), 126
testSubset (checkSubset), 129
testthat, 146
testTibble (checkTibble), 131
testTRUE (checkTRUE), 135
testVector (checkVector), 136
typeof, 29, 53, 126, 131

unique, 86

vname, 10, 11, 13, 16, 19, 22, 27, 29, 31, 34,
37, 41, 46, 49, 51, 52, 56–58, 63, 64,
66–69, 71, 74, 78, 82, 86, 89, 90, 94,
95, 97, 100, 101, 104, 108, 109, 111,
115, 117, 118, 120, 122, 123, 125,
128, 130, 134–136, 139, 144, 145, 147, 148

wf, 5, 149
wl (wf), 149