

Package ‘checkmate’

August 18, 2025

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 2.3.3

URL <https://mllg.github.io/checkmate/>,
<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 (>= 3.0.4), tinytest (>= 1.1.0), tibble

License BSD_3_clause + file LICENSE

VignetteBuilder knitr

RoxygenNote 7.3.2

Collate 'AssertCollection.R' 'allMissing.R' 'anyInfinite.R'
'anyMissing.R' 'anyNaN.R' 'asInteger.R' 'assert.R' 'helper.R'
'makeExpectation.R' 'makeTest.R' 'makeAssertion.R'
'checkAccess.R' 'checkArray.R' 'checkAtomic.R'
'checkAtomicVector.R' 'checkCharacter.R' 'checkChoice.R'
'checkClass.R' 'checkComplex.R' 'checkCount.R'
'checkDataFrame.R' 'checkDataTable.R' 'checkDate.R'
'checkDirectoryExists.R' 'checkDisjunct.R' 'checkDouble.R'
'checkEnvironment.R' 'checkFALSE.R' 'checkFactor.R'

'checkFileExists.R' 'checkFlag.R' 'checkFormula.R'
 'checkFunction.R' 'checkInt.R' 'checkInteger.R'
 'checkIntegerish.R' 'checkList.R' 'checkLogical.R'
 'checkMatrix.R' 'checkMultiClass.R' 'checkNamed.R'
 'checkNames.R' 'checkNull.R' 'checkNumber.R' 'checkNumeric.R'
 'checkOS.R' 'checkPOSIXct.R' 'checkPathForOutput.R'
 'checkPermutation.R' 'checkR6.R' 'checkRaw.R' 'checkScalar.R'
 'checkScalarNA.R' 'checkSetEqual.R' 'checkString.R'
 'checkSubset.R' 'checkTRUE.R' 'checkTibble.R' 'checkVector.R'
 'coalesce.R' 'isIntegerish.R' 'matchArg.R' 'qassert.R'
 'qassertr.R' 'vname.R' 'wfwl.R' 'zzz.R'

Author Michel Lang [cre, aut] (ORCID: <<https://orcid.org/0000-0001-9754-0393>>),
 Bernd Bischl [ctb],
 Dénes Tóth [ctb] (ORCID: <<https://orcid.org/0000-0003-4262-3217>>)

Maintainer Michel Lang <michellang@gmail.com>

Repository CRAN

Date/Publication 2025-08-18 16:30:02 UTC

Contents

checkmate-package	4
allMissing	6
anyInfinite	7
anyNaN	8
asInteger	8
assert	11
AssertCollection	12
checkAccess	13
checkArray	14
checkAtomic	17
checkAtomicVector	20
checkCharacter	23
checkChoice	28
checkClass	30
checkComplex	32
checkCount	36
checkDataFrame	39
checkDataTable	43
checkDate	47
checkDirectoryExists	51
checkDisjunct	53
checkDouble	54
checkEnvironment	59
checkFactor	60
checkFALSE	65
checkFileExists	66

checkFlag	68
checkFormula	70
checkFunction	71
checkInt	73
checkInteger	76
checkIntegerish	80
checkList	85
checkLogical	89
checkMatrix	92
checkMultiClass	97
checkNamed	98
checkNames	99
checkNull	103
checkNumber	104
checkNumeric	107
checkOS	111
checkPathForOutput	112
checkPermutation	114
checkPOSIXct	116
checkR6	119
checkRaw	122
checkScalar	125
checkScalarNA	127
checkSetEqual	128
checkString	130
checkSubset	133
checkTibble	136
checkTRUE	140
checkVector	141
makeAssertion	144
makeExpectation	145
makeTest	147
matchArg	148
qassert	149
qassertr	151
register_test_backend	152
vname	153
wf	153
%??%	154

checkmate-package	<i>checkmate: Fast and Versatile Argument Checks</i>
-------------------	--

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](#)
- [checkPermutation](#)

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)

Author(s)

Maintainer: Michel Lang <michellang@gmail.com> ([ORCID](#))

Other contributors:

- Bernd Bischl <bernd_bischl@gmx.net> [contributor]
- Dénes Tóth <toth.denes@kogentum.hu> ([ORCID](#)) [contributor]

See Also

Useful links:

- <https://mllg.github.io/checkmate/>
- <https://github.com/mllg/checkmate>
- Report bugs at <https://github.com/mllg/checkmate/issues>

allMissing

Check if an object contains missing values

Description

`anyMissing` checks for the presence of at least one missing value, `allMissing` checks for the presence of at least one non-missing value. 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 at least a single non-missing observation present in the `data.frame`, use `all(sapply(df, allMissing))` instead.

Usage

```
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.

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

```

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)
)

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

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 <code>sqrt(.Machine\$double.eps)</code> .
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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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	<i>Combine multiple checks into one assertion</i>
--------	---

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, add = NULL)
```

Arguments

<code>...</code>	[any] List of calls to check functions.
<code>combine</code>	[character(1)] "or" or "and" to combine the check functions with an OR or AND, respectively.
<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See AssertCollection .

Value

Throws an error (or pushes the error message to an [AssertCollection](#) if `add` is not `NULL`) if the checks fail and invisibly returns `TRUE` otherwise.

Examples

```
x = 1:10
assert(checkNull(x), checkInteger(x, any.missing = FALSE))
collection <- makeAssertCollection()
assert(checkChoice(x, c("a", "b")), checkDataFrame(x), add = collection)
collection$getMessage()
```

AssertCollection	<i>Collect multiple assertions</i>
------------------	------------------------------------

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 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

```
makeAssertCollection()
```

```
reportAssertions(collection)
```

Arguments

collection	[AssertCollection] Object of type “AssertCollection” (constructed via <code>makeAssertCollection</code>).
------------	---

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

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

print(coll$isEmpty())
assertNumeric(x, add = coll)
coll$isEmpty()
coll$push("Custom error message")
coll$getMessages()
## Not run:
  reportAssertions(coll)

## End(Not run)
```

checkAccess	<i>Check file system access rights</i>
-------------	--

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 .

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/check_access` return 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")
```

checkArray	<i>Check if an argument is an array</i>
------------	---

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
)
```

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

```
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  
)
```

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

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

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

```

    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). If all values of x are missing, this check is skipped.
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/check_array return a string with the error message. The function expect_array always returns an [expectation](#).

See Also

Other basetypes: [checkAtomic\(\)](#), [checkAtomicVector\(\)](#), [checkCharacter\(\)](#), [checkComplex\(\)](#), [checkDataFrame\(\)](#), [checkDate\(\)](#), [checkDouble\(\)](#), [checkEnvironment\(\)](#), [checkFactor\(\)](#), [checkFormula\(\)](#), [checkFunction\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [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	<i>Check that an argument is an atomic vector</i>
-------------	---

Description

For the definition of “atomic”, see [is.atomic](#).

Note that ‘NULL’ is recognized as a valid atomic value, as in R versions up to version 4.3.x. For details, see <https://stat.ethz.ch/pipermail/r-devel/2023-September/082892.html>.

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

```
assert_atomic(  
  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,
  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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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/check_atomic` return 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\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [checkList\(\)](#), [checkLogical\(\)](#), [checkMatrix\(\)](#), [checkNull\(\)](#), [checkNumeric\(\)](#), [checkPOSIXct\(\)](#), [checkRaw\(\)](#), [checkVector\(\)](#)

Other atomicvector: [checkAtomicVector\(\)](#), [checkVector\(\)](#)

Examples

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

<code>checkAtomicVector</code>	<i>Check that an argument is an atomic vector</i>
--------------------------------	---

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,
    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](#).

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/check_atomic_vector` return 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\(\)](#), [checkFunction\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [checkList\(\)](#), [checkLogical\(\)](#), [checkMatrix\(\)](#), [checkNull\(\)](#), [checkNumeric\(\)](#), [checkPOSIXct\(\)](#), [checkRaw\(\)](#), [checkVector\(\)](#)

Other atomicvector: [checkAtomic\(\)](#), [checkVector\(\)](#)

Examples

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

checkCharacter	<i>Check if an argument is a vector of type character</i>
----------------	---

Description

To check for scalar strings, see [checkString](#).

Usage

```
checkCharacter(
  x,
  n.chars = NULL,
  min.chars = NULL,
  max.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,  
    typed.missing = FALSE,  
    null.ok = FALSE  
)
```

```
check_character(  
  x,  
  n.chars = NULL,  
  min.chars = NULL,  
  max.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,  
  typed.missing = FALSE,  
  null.ok = FALSE  
)
```

```
assertCharacter(  
  x,  
  n.chars = NULL,  
  min.chars = NULL,  
  max.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,  
  typed.missing = FALSE,  
  null.ok = FALSE,  
  .var.name = vname(x),  
  add = NULL  
)
```



```
assert_character(  
  x,  
  n.chars = NULL,  
  min.chars = NULL,  
  max.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,  
  typed.missing = FALSE,  
  null.ok = FALSE,  
  .var.name = vname(x),  
  add = NULL  
)
```

```
testCharacter(  
  x,  
  n.chars = NULL,  
  min.chars = NULL,  
  max.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,  
  typed.missing = FALSE,  
  null.ok = FALSE  
)
```

```
test_character(  
  x,  
  n.chars = NULL,  
  min.chars = NULL,  
  max.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,
    typed.missing = FALSE,
    null.ok = FALSE
  )

expect_character(
  x,
  n.chars = NULL,
  min.chars = NULL,
  max.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,
  typed.missing = FALSE,
  null.ok = FALSE,
  info = NULL,
  label = vname(x)
)

```

Arguments

<code>x</code>	[any] Object to check.
<code>n.chars</code>	[integer(1)] Exact number of characters for each element of <code>x</code> .
<code>min.chars</code>	[integer(1)] Minimum number of characters for each element of <code>x</code> .
<code>max.chars</code>	[integer(1)] Maximum number of characters for each element of <code>x</code> .

pattern	[character(1)] 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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.
typed.missing	[logical(1)] If set to FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, NA_character_ or NA_character_) as well as empty vectors are allowed while type-checking atomic input. Set to TRUE to enable strict type checking.
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 `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/check_character` return a string with the error message. The function `expect_character` always returns an [expectation](#).

See Also

Other basetypes: [checkArray\(\)](#), [checkAtomic\(\)](#), [checkAtomicVector\(\)](#), [checkComplex\(\)](#), [checkDataFrame\(\)](#), [checkDate\(\)](#), [checkDouble\(\)](#), [checkEnvironment\(\)](#), [checkFactor\(\)](#), [checkFormula\(\)](#), [checkFunction\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [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")
```

checkChoice	<i>Check if an object is an element of a given set</i>
-------------	--

Description

Check if an object is an element of a given set

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 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 `assertChoice/assert_choice` return `x` invisibly, whereas `checkChoice/check_choice` and `testChoice/test_choice` return `TRUE`. If the check is not successful, `assertChoice/assert_choice` throws an error message, `testChoice/test_choice` returns `FALSE`, and `checkChoice/check_choice` return a string with the error message. The function `expect_choice` 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: [checkDisjunct\(\)](#), [checkPermutation\(\)](#), [checkSetEqual\(\)](#), [checkSubset\(\)](#)

Examples

```
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))
```

checkClass

Check the class membership of an argument

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 inherits . x must inherit from all specified classes.
ordered	[logical(1)] Expect x to be specialized in provided order. 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 `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/check_class` return 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)
# is x 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,  
    typed.missing = FALSE,  
    null.ok = FALSE  
)
```

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

```
assertComplex(  
  x,  
  any.missing = TRUE,  
  all.missing = TRUE,  
  len = NULL,  
  min.len = NULL,  
  max.len = NULL,  
  unique = FALSE,  
  names = NULL,  
  typed.missing = FALSE,  
  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,  
  typed.missing = FALSE,  
  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,  
  typed.missing = FALSE,  
  null.ok = FALSE  
)  
  
test_complex(  
  x,  
  any.missing = TRUE,  
  all.missing = TRUE,  
  len = NULL,  
  min.len = NULL,  
  max.len = NULL,  
  unique = FALSE,  
  names = NULL,  
  typed.missing = FALSE,  
  null.ok = FALSE  
)  
  
expect_complex(  
  x,  
  any.missing = TRUE,  
  all.missing = TRUE,  
  len = NULL,  
  min.len = NULL,  
  max.len = NULL,  
  unique = FALSE,  
  names = NULL,  
  typed.missing = FALSE,  
  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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.
typed.missing	[logical(1)] If set to FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, NA_character_ or NA_character_) as well as empty vectors are allowed while type-checking atomic input. Set to TRUE to enable strict type checking.
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 `assertComplex/assert_complex` return x invisibly, whereas `checkComplex/check_complex` and `testComplex/test_complex` re-

turn TRUE. If the check is not successful, `assertComplex/assert_complex` throws an error message, `testComplex/test_complex` returns FALSE, and `checkComplex/check_complex` return a string with the error message. The function `expect_complex` always returns an [expectation](#).

See Also

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

Examples

```
testComplex(1)
testComplex(1+1i)
```

checkCount	<i>Check if an argument is a count</i>
------------	--

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.
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 <code>sqrt(.Machine\$double.eps)</code> .
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/check_count` return 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\(\)](#), [checkScalar\(\)](#), [checkScalarNA\(\)](#), [checkString\(\)](#)

Examples

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

checkDataFrame	<i>Check if an argument is a data frame</i>
----------------	---

Description

Check if an argument is a data frame

Usage

```
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  
)
```

```
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  
)
```

```
assertDataFrame(  
  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_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
)
```



```

)

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/check_data_frame` return 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\(\)](#), [checkAtomic\(\)](#), [checkAtomicVector\(\)](#), [checkCharacter\(\)](#), [checkComplex\(\)](#), [checkDate\(\)](#), [checkDouble\(\)](#), [checkEnvironment\(\)](#), [checkFactor\(\)](#), [checkFormula\(\)](#), [checkFunction\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [checkList\(\)](#), [checkLogical\(\)](#), [checkMatrix\(\)](#), [checkNull\(\)](#), [checkNumeric\(\)](#), [checkPOSIXct\(\)](#), [checkRaw\(\)](#), [checkVector\(\)](#)

Examples

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

checkDataTable	<i>Check if an argument is a data table</i>
----------------	---

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,
```

```
    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
  )

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,
```

```

    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.
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” “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 <code>character(0)</code> (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 .
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 `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/check_data_table` return a string with the error message. The function `expect_data_table` always returns an [expectation](#).

See Also

Other compound: [checkArray\(\)](#), [checkDataFrame\(\)](#), [checkMatrix\(\)](#), [checkTibble\(\)](#)

Examples

```
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)
```

checkDate	<i>Check that an argument is a Date</i>
-----------	---

Description

Checks that an object is of class [Date](#).

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,  
  label = 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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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/check_atomic` return a string with the error message. The function `expect_atomic` always returns an [expectation](#).

See Also

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

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.

<code>access</code>	[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.
<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See AssertCollection .
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that .
<code>label</code>	[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/check_directory_exists` return 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	<i>Check if an argument is disjunct from a given set</i>
---------------	--

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 .

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/check_disjunct` return 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\(\)](#), [checkPermutation\(\)](#), [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,
```

```
    typed.missing = FALSE,  
    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,  
  typed.missing = FALSE,  
  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,  
  typed.missing = FALSE,  
  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,  
    sorted = FALSE,  
    names = NULL,  
    typed.missing = FALSE,  
    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,  
  typed.missing = FALSE,  
  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,  
  typed.missing = FALSE,  
  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,
    unique = FALSE,
    sorted = FALSE,
    names = NULL,
    typed.missing = FALSE,
    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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.

typed.missing	[logical(1)] If set to FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, NA_character_ or NA_character_) as well as empty vectors are allowed while type-checking atomic input. Set to TRUE to enable strict type checking.
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 `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/check_double` return a string with the error message. The function `expect_double` always returns an [expectation](#).

See Also

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

Examples

```
testDouble(1)
testDouble(1L)
testDouble(1, min.len = 1, lower = 0)
```

checkEnvironment	<i>Check if an argument is an environment</i>
------------------	---

Description

Check if an argument is an environment

Usage

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

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

assertEnvironment(
  x,
  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 .

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/check_environment` return a string with the error message. The function `expect_environment` always returns an [expectation](#).

See Also

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

Examples

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

checkFactor	<i>Check if an argument is a factor</i>
-------------	---

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,  
  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,
```

```

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 un-ordered 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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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).
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/check_factor` return a string with the error message. The function `expect_factor` always returns an [expectation](#).

See Also

Other basetypes: [checkArray\(\)](#), [checkAtomic\(\)](#), [checkAtomicVector\(\)](#), [checkCharacter\(\)](#), [checkComplex\(\)](#), [checkDataFrame\(\)](#), [checkDate\(\)](#), [checkDouble\(\)](#), [checkEnvironment\(\)](#), [checkFormula\(\)](#), [checkFunction\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [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	<i>Check if an argument is FALSE</i>
------------	--------------------------------------

Description

Simply checks if an argument is `FALSE`.

Usage

```
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 <code>FALSE</code> .
.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./check_false.` return a string with the error message. The function `expect_false.` always returns an [expectation](#).

Examples

```
testFALSE(FALSE)
testFALSE(TRUE)
```

checkFileExists	<i>Check existence and access rights of files</i>
-----------------	---

Description

Check existence and access rights of files

Usage

```
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)

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/check_file_exists` return 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	<i>Check if an argument is a flag</i>
-----------	---------------------------------------

Description

A flag is defined as single logical value.

Usage

```
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)

expect_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 .
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/check_flag` return a string with the error message. The function `expect_flag` always returns an [expectation](#).

See Also

Other scalars: [checkCount\(\)](#), [checkInt\(\)](#), [checkNumber\(\)](#), [checkScalar\(\)](#), [checkScalarNA\(\)](#), [checkString\(\)](#)

Examples

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

checkFormula	<i>Check if an argument is a formula</i>
--------------	--

Description

Check if an argument is a formula

Usage

```
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/check_formula` return a string with the error message. The function `expect_formula` always returns an [expectation](#).

See Also

Other basetypes: [checkArray\(\)](#), [checkAtomic\(\)](#), [checkAtomicVector\(\)](#), [checkCharacter\(\)](#), [checkComplex\(\)](#), [checkDataFrame\(\)](#), [checkDate\(\)](#), [checkDouble\(\)](#), [checkEnvironment\(\)](#), [checkFactor\(\)](#), [checkFunction\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [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)
)

```

Arguments

x	[any] Object to check.
args	[character] Expected formal arguments. Checks that a function has no arguments if set to <code>character(0)</code> . Default is <code>NULL</code> (no check).
ordered	[logical(1)] Flag whether the arguments provided in <code>args</code> must be the first <code>length(args)</code> arguments of the function in the specified order. Default is <code>FALSE</code> .
nargs	[integer(1)] Required number of arguments, without <code>...</code> . Default is <code>NULL</code> (no check).
null.ok	[logical(1)] If set to <code>TRUE</code> , <code>x</code> may also be <code>NULL</code> . In this case only a type check of <code>x</code> 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 <code>testthat</code> 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/check_function` return a string with the error message. The function `expect_function` always returns an [expectation](#).

See Also

Other basetypes: [checkArray\(\)](#), [checkAtomic\(\)](#), [checkAtomicVector\(\)](#), [checkCharacter\(\)](#), [checkComplex\(\)](#), [checkDataFrame\(\)](#), [checkDate\(\)](#), [checkDouble\(\)](#), [checkEnvironment\(\)](#), [checkFactor\(\)](#), [checkFormula\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [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

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

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

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  
)  
  
assert_int(  
  x,  
  na.ok = FALSE,  
  lower = -Inf,  
  upper = Inf,  
  tol = sqrt(.Machine$double.eps),  
  null.ok = FALSE,  
  coerce = FALSE,  
  .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 <code>sqrt(.Machine\$double.eps)</code> .
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 `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/check_int` return 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\(\)](#), [checkScalar\(\)](#), [checkScalarNA\(\)](#), [checkString\(\)](#)

Examples

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

checkInteger	<i>Check if an argument is vector of type integer</i>
--------------	---

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,
  typed.missing = FALSE,
  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,  
    typed.missing = FALSE,  
    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,  
  typed.missing = FALSE,  
  null.ok = FALSE,  
  .var.name = vname(x),  
  add = NULL  
)
```

```
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,  
  typed.missing = FALSE,  
  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,  
typed.missing = FALSE,  
null.ok = FALSE  
)
```

```
test_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,  
  typed.missing = FALSE,  
  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,  
  typed.missing = FALSE,  
  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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.
typed.missing	[logical(1)] If set to FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, NA_character_ or NA_character_) as well as empty vectors are allowed while type-checking atomic input. Set to TRUE to enable strict type checking.
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/check_integer` return a string with the error message. The function `expect_integer` always returns an [expectation](#).

See Also

[asInteger](#)

Other basetypes: [checkArray\(\)](#), [checkAtomic\(\)](#), [checkAtomicVector\(\)](#), [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 sufficiently close to an integer w.r.t. a numeric tolerance 'tol'.

Usage

```
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,
    typed.missing = FALSE,
    null.ok = FALSE
)

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,
  typed.missing = FALSE,
  null.ok = FALSE
)

assertIntegerish(
  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,
  typed.missing = FALSE,
  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,
typed.missing = FALSE,
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,
  typed.missing = FALSE,
  null.ok = FALSE
)

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,
  typed.missing = FALSE,
  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,
  typed.missing = FALSE,
  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 <code>sqrt(.Machine\$double.eps)</code> .
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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.
typed.missing	[logical(1)] If set to FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, NA_character_ or NA_character_) as well as empty vectors are allowed while type-checking atomic input. Set to TRUE to enable strict type checking.
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/check_integerish` return a string with the error message. The function `expect_integerish` always returns an [expectation](#).

Note

To convert from integerish to integer, use [asInteger](#).

See Also

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

Examples

```
testIntegerish(1L)
testIntegerish(1.)
testIntegerish(1:2, lower = 1L, upper = 2L, any.missing = FALSE)
```

checkList	<i>Check if an argument is a list</i>
-----------	---------------------------------------

Description

Check if an argument is a list

Usage

```
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
)
```

```
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
)
```

```
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(  
  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 <code>character(0)</code> (no check).
any.missing	[logical(1)] Are vectors with missing values allowed? Default is TRUE.
all.missing	[logical(1)] Are vectors with no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.

<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in <code>vname</code> .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See <code>AssertCollection</code> .
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See <code>expect_that</code> .
<code>label</code>	[character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in <code>vname</code> .

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/check_list` return a string with the error message. The function `expect_list` always returns an `expectation`.

Note

Contrary to R's `is.list`, objects of type `data.frame` and `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()`, `checkAtomic()`, `checkAtomicVector()`, `checkCharacter()`, `checkComplex()`, `checkDataFrame()`, `checkDate()`, `checkDouble()`, `checkEnvironment()`, `checkFactor()`, `checkFormula()`, `checkFunction()`, `checkInteger()`, `checkIntegerish()`, `checkLogical()`, `checkMatrix()`, `checkNull()`, `checkNumeric()`, `checkPOSIXct()`, `checkRaw()`, `checkVector()`

Examples

```
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	<i>Check if an argument is a vector of type logical</i>
--------------	---

Description

Check if an argument is a vector of type logical

Usage

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

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

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

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

  expect_logical(
    x,
    any.missing = TRUE,
    all.missing = TRUE,
```

```

    len = NULL,
    min.len = NULL,
    max.len = NULL,
    unique = FALSE,
    names = NULL,
    typed.missing = FALSE,
    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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.
typed.missing	[logical(1)] If set to FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, NA_character_ or NA_character_) as well as empty vectors are allowed while type-checking atomic input. Set to TRUE to enable strict type checking.
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 `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 message, `testLogical/test_logical` returns `FALSE`, and `checkLogical/check_logical` return a string with the error message. The function `expect_logical` always returns an [expectation](#).

See Also

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

Examples

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

checkMatrix	<i>Check if an argument is a matrix</i>
-------------	---

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,
```

```
    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,
```

```
    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,
```

```

    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). If all values of x are missing, this check is skipped.
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 .
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/check_matrix` return a string with the error message. The function `expect_matrix` always returns an [expectation](#).

See Also

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

Other compound: [checkArray\(\)](#), [checkDataFrame\(\)](#), [checkDataTable\(\)](#), [checkTibble\(\)](#)

Examples

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

checkMultiClass	<i>Check the class membership of an argument</i>
-----------------	--

Description

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` return `TRUE`. If the check is not successful, `assertMultiClass/assert_multi_class` throws an error message, `testMultiClass/test_multi_class` returns `FALSE`, and `checkMultiClass/check_multi_class` return 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	<i>Check if an argument is named</i>
------------	--------------------------------------

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.
.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 `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/check_named` return a string with the error message. The function `expect_named` always returns an [expectation](#).

Note

These function are deprecated and will be removed in a future version. Please use [checkNames](#) instead.

See Also

Other attributes: [checkClass\(\)](#), [checkMultiClass\(\)](#), [checkNames\(\)](#)

Examples

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

checkNames

Check names to comply to specific rules

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"  
)
```

```
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"
)
```

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

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

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._]*$`”.

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

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

<code>subset.of</code>	[character] Names provided in <code>x</code> must be subset of the set <code>subset.of</code> .
<code>must.include</code>	[character] Names provided in <code>x</code> must be a superset of the set <code>must.include</code> .
<code>permutation.of</code>	[character] Names provided in <code>x</code> must be a permutation of the set <code>permutation.of</code> . Duplicated names in <code>permutation.of</code> are stripped out and duplicated names in <code>x</code> thus lead to a failed check. Use this argument instead of <code>identical.to</code> if the order of the names is not relevant.
<code>identical.to</code>	[character] Names provided in <code>x</code> must be identical to the vector <code>identical.to</code> . Use this argument instead of <code>permutation.of</code> if the order of the names is relevant.
<code>disjunct.from</code>	[character] Names provided in <code>x</code> must may not be present in the vector <code>disjunct.from</code> .
<code>what</code>	[character(1)] Type of name vector to check, e.g. “names” (default), “colnames” or “rownames”.
<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See AssertCollection .
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that .
<code>label</code>	[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 `assertNames/assert_names` return `x` invisibly, whereas `checkNames/check_names` and `testNames/test_names` return `TRUE`. If the check is not successful, `assertNames/assert_names` throws an error message, `testNames/test_names` returns `FALSE`, and `checkNames/check_names` return a string with the error message. The function `expect_names` always returns an [expectation](#).

See Also

Other attributes: [checkClass\(\)](#), [checkMultiClass\(\)](#), [checkNamed\(\)](#)

Examples

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

cn = c("Species", "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")
assertNames(names(iris), permutation.of = cn)
```

checkNull	<i>Check if an argument is NULL</i>
-----------	-------------------------------------

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/check_null` return a string with the error message. The function `expect_null` always returns an [expectation](#).

See Also

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

Examples

```
testNull(NULL)
testNull(1)
```

checkNumber

Check if an argument is a single numeric value

Description

Check if an argument is a single numeric value

Usage

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

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

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



```
)

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

testNumber(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  null.ok = FALSE
)

test_number(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  null.ok = FALSE
)

expect_number(
  x,
  na.ok = FALSE,
  lower = -Inf,
  upper = Inf,
  finite = 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.

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/check_number` return a string with the error message. The function `expect_number` always returns an [expectation](#).

See Also

Other scalars: [checkCount\(\)](#), [checkFlag\(\)](#), [checkInt\(\)](#), [checkScalar\(\)](#), [checkScalarNA\(\)](#), [checkString\(\)](#)

Examples

```
testNumber(1)
testNumber(1:2)
```

checkNumeric	<i>Check that an argument is a vector of type numeric</i>
--------------	---

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,  
  typed.missing = FALSE,  
  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,  
  typed.missing = FALSE,  
  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,  
typed.missing = FALSE,  
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,  
  typed.missing = FALSE,  
  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,
```

```
    typed.missing = FALSE,
    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,
  sorted = FALSE,
  names = NULL,
  typed.missing = FALSE,
  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,
  typed.missing = FALSE,
  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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.
typed.missing	[logical(1)] If set to FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, NA_character_ or NA_character_) as well as empty vectors are allowed while type-checking atomic input. Set to TRUE to enable strict type checking.
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/check_numeric` return a string with the error message. The function `expect_numeric` always returns an [expectation](#).

See Also

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

Examples

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

checkOS	<i>Check the operating system</i>
---------	-----------------------------------

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] 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/check_os` return a string with the error message. The function `expect_os` always returns an [expectation](#).

Examples

```
testOS("linux")
```

checkPathForOutput	<i>Check if a path is suited for creating an output file</i>
--------------------	--

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,
        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 is both readable and writable. Default is FALSE.
extension	[character(1)] Extension of the file, e.g. “txt” 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/check_path_for_output` return a string with the error message. The function `expect_path_for_output` always returns an [expectation](#).

See Also

Other filesystem: [checkAccess\(\)](#), [checkDirectoryExists\(\)](#), [checkFileExists\(\)](#)

Examples

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

checkPermutation	<i>Check if the arguments are permutations of each other.</i>
------------------	---

Description

In contrast to [checkSetEqual](#), the function tests for a true permutation of the two vectors and also considers duplicated values. Missing values are being treated as actual values by default. Does not work on raw values.

Usage

```
checkPermutation(x, y, na.ok = TRUE)

check_permutation(x, y, na.ok = TRUE)

assertPermutation(x, y, na.ok = TRUE, .var.name = vname(x), add = NULL)

assert_permutation(x, y, na.ok = TRUE, .var.name = vname(x), add = NULL)

testPermutation(x, y, na.ok = TRUE)

test_permutation(x, y, na.ok = TRUE)

expect_permutation(x, y, na.ok = TRUE, info = NULL, label = vname(x))
```

Arguments

`x` [any]
Object to check.

<code>y</code>	[atomic] Vector to compare with. Atomic vector of type other than raw.
<code>na.ok</code>	[logical(1)] Are missing values allowed? Default is TRUE.
<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See AssertCollection .
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that .
<code>label</code>	[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 `assertPermutation/assert_permutation` return `x` invisibly, whereas `checkPermutation/check_permutation` and `testPermutation/test_permutation` return TRUE. If the check is not successful, `assertPermutation/assert_permutation` throws an error message, `testPermutation/test_permutation` returns FALSE, and `checkPermutation/check_permutation` return a string with the error message. The function `expect_permutation` 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\(\)](#), [checkSubset\(\)](#)

Examples

```
testPermutation(letters[1:2], letters[2:1])
testPermutation(letters[c(1, 1, 2)], letters[1:2])
testPermutation(c(NA, 1, 2), c(1, 2, NA))
testPermutation(c(NA, 1, 2), c(1, 2, NA), na.ok = FALSE)
```

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

Description

Checks that an object is of class `POSIXct`.

Usage

```
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  
)
```

```
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  
)
```

```
assertPOSIXct(  
  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  
  )
```

```
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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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.

<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See AssertCollection .
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that .
<code>label</code>	[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/check_atomic` return a string with the error message. The function `expect_atomic` always returns an [expectation](#).

See Also

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

checkR6

Check if an argument is an R6 class

Description

Check if an argument is an R6 class

Usage

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

```
check_r6(  
  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 .

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/check_class` return a string with the error message. The function `expect_class` always returns an [expectation](#).

See Also

Other classes: [checkClass\(\)](#), [checkMultiClass\(\)](#)

Examples

```
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")
```

checkRaw	<i>Check if an argument is a raw vector</i>
----------	---

Description

Check if an argument is a raw vector

Usage

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

check_raw(
  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,
```

```

    len = NULL,
    min.len = NULL,
    max.len = NULL,
    names = NULL,
    null.ok = FALSE,
    info = NULL,
    label = vname(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/check_raw` return a string with the error message. The function `expect_raw` always returns an [expectation](#).

See Also

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

Examples

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

checkScalar	<i>Check if an argument is a single atomic value</i>
-------------	--

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))
```

Arguments

<code>x</code>	[any] Object to check.
<code>na.ok</code>	[logical(1)] Are missing values allowed? Default is FALSE.
<code>null.ok</code>	[logical(1)] If set to TRUE, <code>x</code> may also be NULL. In this case only a type check of <code>x</code> is performed, all additional checks are disabled.
<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See AssertCollection .
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that .
<code>label</code>	[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/check_scalar` return 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	<i>Check if an argument is a single missing value</i>
---------------	---

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 .

Value

Depending on the function prefix: If the check is successful, the functions `assertScalarNA/assert_scalar_na` return `x` invisibly, whereas `checkScalarNA/check_scalar_na` and `testScalarNA/test_scalar_na` return `TRUE`. If the check is not successful, `assertScalarNA/assert_scalar_na` throws an error message, `testScalarNA/test_scalar_na` returns `FALSE`, and `checkScalarNA/check_scalar_na` return a string with the error message. The function `expect_scalar_na` always returns an [expectation](#).

See Also

Other scalars: [checkCount\(\)](#), [checkFlag\(\)](#), [checkInt\(\)](#), [checkNumber\(\)](#), [checkScalar\(\)](#), [checkString\(\)](#)

Examples

```
testScalarNA(1)
testScalarNA(NA_real_)
testScalarNA(rep(NA, 2))
```

checkSetEqual

Check if an argument is equal to a given set

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
)
```



```
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/check_subset` return 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\(\)](#), [checkPermutation\(\)](#), [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))
```

checkString	<i>Check if an argument is a string</i>
-------------	---

Description

A string is defined as a scalar character vector. To check for vectors of arbitrary length, see [checkCharacter](#).

Usage

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

check_string(
  x,
  na.ok = FALSE,
  n.chars = NULL,
  min.chars = NULL,
  max.chars = NULL,
  pattern = NULL,
```

```
    fixed = NULL,  
    ignore.case = FALSE,  
    null.ok = FALSE  
  )  
  
  assertString(  
    x,  
    na.ok = FALSE,  
    n.chars = NULL,  
    min.chars = NULL,  
    max.chars = NULL,  
    pattern = NULL,  
    fixed = NULL,  
    ignore.case = FALSE,  
    null.ok = FALSE,  
    .var.name = vname(x),  
    add = NULL  
  )  
  
  assert_string(  
    x,  
    na.ok = FALSE,  
    n.chars = NULL,  
    min.chars = NULL,  
    max.chars = NULL,  
    pattern = NULL,  
    fixed = NULL,  
    ignore.case = FALSE,  
    null.ok = FALSE,  
    .var.name = vname(x),  
    add = NULL  
  )  
  
  testString(  
    x,  
    na.ok = FALSE,  
    n.chars = NULL,  
    min.chars = NULL,  
    max.chars = NULL,  
    pattern = NULL,  
    fixed = NULL,  
    ignore.case = FALSE,  
    null.ok = FALSE  
  )  
  
  test_string(  
    x,  
    na.ok = FALSE,
```

```

    n.chars = NULL,
    min.chars = NULL,
    max.chars = NULL,
    pattern = NULL,
    fixed = NULL,
    ignore.case = FALSE,
    null.ok = FALSE
  )

  expect_string(
    x,
    na.ok = FALSE,
    n.chars = NULL,
    min.chars = NULL,
    max.chars = NULL,
    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.
n.chars	[integer(1)] Exact number of characters for each element of x.
min.chars	[integer(1)] Minimum number of characters for each element of x.
max.chars	[integer(1)] Maximum number of characters for each element of x.
pattern	[character(1)] 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.

<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See AssertCollection .
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that .
<code>label</code>	[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 `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/check_string` return a string with the error message. The function `expect_string` always returns an [expectation](#).

See Also

Other scalars: [checkCount\(\)](#), [checkFlag\(\)](#), [checkInt\(\)](#), [checkNumber\(\)](#), [checkScalar\(\)](#), [checkScalarNA\(\)](#)

Examples

```
testString("a")
testString(letters)
```

checkSubset	<i>Check if an argument is a subset of a given set</i>
-------------	--

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 fast-

match 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.

<code>.var.name</code>	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in <code>vname</code> .
<code>add</code>	[AssertCollection] Collection to store assertion messages. See <code>AssertCollection</code> .
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See <code>expect_that</code> .
<code>label</code>	[character(1)] Name of the checked object to print in messages. Defaults to the heuristic implemented in <code>vname</code> .

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/check_subset` return 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()`, `checkPermutation()`, `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	<i>Check if an argument is a tibble</i>
-------------	---

Description

Check if an argument is a tibble

Usage

```
checkTibble(  
  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  
)
```

```
check_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  
)
```

```
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
  )

  expect_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,
    label = vname(x)
  )

```

Arguments

<code>x</code>	[any] Object to check.
<code>types</code>	[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 <code>character(0)</code> (no check).
<code>any.missing</code>	[logical(1)] Are missing values allowed? Default is TRUE.
<code>all.missing</code>	[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 .
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 `assertTibble/assert_tibble` return `x` invisibly, whereas `checkTibble/check_tibble` and `testTibble/test_tibble` return `TRUE`. If the check is not successful, `assertTibble/assert_tibble` throws an error message, `testTibble/test_tibble` returns `FALSE`, and `checkTibble/check_tibble` return a string with the error message. The function `expect_tibble` always returns an [expectation](#).

See Also

Other compound: [checkArray\(\)](#), [checkDataFrame\(\)](#), [checkDataTable\(\)](#), [checkMatrix\(\)](#)

Examples

```
library(tibble)
x = as_tibble(iris)
testTibble(x)
testTibble(x, nrow = 150, any.missing = FALSE)
```

checkTRUE

*Check if an argument is TRUE***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)
```

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./check_true.` return a string with the error message. The function `expect_true.` always returns an [expectation](#).

Examples

```
testTRUE(TRUE)
testTRUE(FALSE)
```

checkVector	<i>Check if an argument is a vector</i>
-------------	---

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
)
```

```
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 no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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/check_vector` return a string with the error message. The function `expect_vector` always returns an [expectation](#).

See Also

Other basetypes: [checkArray\(\)](#), [checkAtomic\(\)](#), [checkAtomicVector\(\)](#), [checkCharacter\(\)](#), [checkComplex\(\)](#), [checkDataFrame\(\)](#), [checkDate\(\)](#), [checkDouble\(\)](#), [checkEnvironment\(\)](#), [checkFactor\(\)](#),

[checkFormula\(\)](#), [checkFunction\(\)](#), [checkInteger\(\)](#), [checkIntegerish\(\)](#), [checkList\(\)](#), [checkLogical\(\)](#), [checkMatrix\(\)](#), [checkNull\(\)](#), [checkNumeric\(\)](#), [checkPOSIXct\(\)](#), [checkRaw\(\)](#)

Other atomicvector: [checkAtomic\(\)](#), [checkAtomicVector\(\)](#)

Examples

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

makeAssertion	<i>Turn a Check into an Assertion</i>
---------------	---------------------------------------

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()
)
```

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	<i>Turn a Check into an Expectation</i>
-----------------	---

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

```
makeExpectation(x, res, info, label)

makeExpectationFunction(
  check.fun,
  c.fun = NULL,
  use.namespace = FALSE,
  env = parent.frame()
)
```

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.
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.
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

```
# 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)
```

makeTest	<i>Turn a Check into a Test</i>
----------	---------------------------------

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

```
makeTest(res)

makeTestFunction(check.fun, c.fun = NULL, env = parent.frame())
```

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

```
# 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

```
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

```
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

```
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.
[Ø]	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
[Ø-9]+	exact length specified as integer.

Preceding the exact length with one of the comparison operators `==`, `<`, `<=`, `>=` 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 `all(x >= 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 [Ø,] contains `Inf`, while [Ø,) does not. The same holds for the left (lower) boundary and `-Inf`. E.g., the rule “N1()” checks for a single finite numeric which is not NA, while “N1[]” allows `-Inf`.

Value

`qassert` throws an R exception if object `x` does not comply to at least one of the rules and returns the tested object invisibly otherwise. `qtest` behaves the same way but returns `FALSE` if none of the rules comply. `qexpect` is intended to be inside the unit test framework [testthat](#) and returns an [expectation](#).

Note

The functions are inspired by the blog post of Bogumił Kamiński: <http://rsnippets.blogspot.de/2013/06/testing-function-arguments-in-gnu-r.html>. The implementation is mostly written in C to minimize the overhead.

See Also

[qtestr](#) and [qassertr](#) for efficient checks of list elements and data frame columns.

Examples

```
# 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 >= 0 and < Inf
qtest(1:3, "I+[0,)")

# either an empty list or a character vector with <=5 elements
qtest(1, c("l0", "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

<code>.var.name</code>	[character(1)] Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname .
<code>depth</code>	[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.
<code>info</code>	[character(1)] Extra information to be included in the message for the testthat reporter. See expect_that .
<code>label</code>	[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	<i>Lookup a variable name</i>
-------	-------------------------------

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.

wf	<i>Get the index of the first/last TRUE</i>
----	---

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))
wl(c(FALSE, FALSE))
wf(NA)
```

???	<i>Coalesce operator</i>
-----	--------------------------

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))])
```

Index

- * **CustomConstructors**
 - makeAssertion, 144
 - makeExpectation, 145
 - makeTest, 147
- * **atomicvector**
 - checkAtomic, 17
 - checkAtomicVector, 20
 - checkVector, 141
- * **attributes**
 - checkClass, 30
 - checkMultiClass, 97
 - checkNamed, 98
 - checkNames, 99
- * **basetypes**
 - checkArray, 14
 - checkAtomic, 17
 - checkAtomicVector, 20
 - checkCharacter, 23
 - checkComplex, 32
 - checkDataFrame, 39
 - checkDate, 47
 - checkDouble, 54
 - checkEnvironment, 59
 - checkFactor, 60
 - checkFormula, 70
 - checkFunction, 71
 - checkInteger, 76
 - checkIntegerish, 80
 - checkList, 85
 - checkLogical, 89
 - checkMatrix, 92
 - checkNull, 103
 - checkNumeric, 107
 - checkPOSIXct, 116
 - checkRaw, 122
 - checkVector, 141
- * **classes**
 - checkClass, 30
 - checkMultiClass, 97
 - checkR6, 119
- * **compound**
 - checkArray, 14
 - checkDataFrame, 39
 - checkDataTable, 43
 - checkMatrix, 92
 - checkTibble, 136
- * **filesystem**
 - checkAccess, 13
 - checkDirectoryExists, 51
 - checkFileExists, 66
 - checkPathForOutput, 112
- * **scalars**
 - checkCount, 36
 - checkFlag, 68
 - checkInt, 73
 - checkNumber, 104
 - checkScalar, 125
 - checkScalarNA, 127
 - checkString, 130
- * **set**
 - checkChoice, 28
 - checkDisjunct, 53
 - checkPermutation, 114
 - checkSetEqual, 128
 - checkSubset, 133
- .Call, 145–147
- %%?, 154
- allMissing, 6, 6
- anyInfinite, 7
- anyMissing, 6, 88
- anyMissing(allMissing), 6
- anyNaN, 6, 8
- asCount, 5, 38
- asCount(asInteger), 8
- asInt, 5, 76
- asInt(asInteger), 8
- asInteger, 5, 8, 80, 84
- assert, 6, 11

- `assert_access` (`checkAccess`), 13
- `assert_array` (`checkArray`), 14
- `assert_atomic` (`checkAtomic`), 17
- `assert_atomic_vector`
 (`checkAtomicVector`), 20
- `assert_character` (`checkCharacter`), 23
- `assert_choice` (`checkChoice`), 28
- `assert_class` (`checkClass`), 30
- `assert_complex` (`checkComplex`), 32
- `assert_count` (`checkCount`), 36
- `assert_data_frame` (`checkDataFrame`), 39
- `assert_data_table` (`checkDataTable`), 43
- `assert_date` (`checkDate`), 47
- `assert_directory`
 (`checkDirectoryExists`), 51
- `assert_directory_exists`
 (`checkDirectoryExists`), 51
- `assert_disjunct` (`checkDisjunct`), 53
- `assert_double` (`checkDouble`), 54
- `assert_environment` (`checkEnvironment`),
 59
- `assert_factor` (`checkFactor`), 60
- `assert_false` (`checkFALSE`), 65
- `assert_file` (`checkFileExists`), 66
- `assert_file_exists` (`checkFileExists`), 66
- `assert_flag` (`checkFlag`), 68
- `assert_formula` (`checkFormula`), 70
- `assert_function` (`checkFunction`), 71
- `assert_int` (`checkInt`), 73
- `assert_integer` (`checkInteger`), 76
- `assert_integerish` (`checkIntegerish`), 80
- `assert_list` (`checkList`), 85
- `assert_logical` (`checkLogical`), 89
- `assert_matrix` (`checkMatrix`), 92
- `assert_multi_class` (`checkMultiClass`), 97
- `assert_named` (`checkNamed`), 98
- `assert_names` (`checkNames`), 99
- `assert_null` (`checkNull`), 103
- `assert_number` (`checkNumber`), 104
- `assert_numeric` (`checkNumeric`), 107
- `assert_os` (`checkOS`), 111
- `assert_path_for_output`
 (`checkPathForOutput`), 112
- `assert_permutation` (`checkPermutation`),
 114
- `assert_posixct` (`checkPOSIXct`), 116
- `assert_r6` (`checkR6`), 119
- `assert_raw` (`checkRaw`), 122
- `assert_scalar` (`checkScalar`), 125
- `assert_scalar_na` (`checkScalarNA`), 127
- `assert_set_equal` (`checkSetEqual`), 128
- `assert_string` (`checkString`), 130
- `assert_subset` (`checkSubset`), 133
- `assert_tibble` (`checkTibble`), 136
- `assert_true` (`checkTRUE`), 140
- `assert_vector` (`checkVector`), 141
- `assertAccess` (`checkAccess`), 13
- `assertArray` (`checkArray`), 14
- `assertAtomic` (`checkAtomic`), 17
- `assertAtomicVector` (`checkAtomicVector`),
 20
- `assertCharacter` (`checkCharacter`), 23
- `assertChoice` (`checkChoice`), 28
- `assertClass` (`checkClass`), 30
- `AssertCollection`, 11, 12, 13, 16, 19, 22, 27,
 29, 31, 35, 38, 42, 47, 50, 52, 53, 58,
 60, 64, 65, 67, 69, 70, 72, 75, 79, 84,
 88, 91, 96, 97, 99, 102, 103, 106,
 110, 112, 113, 115, 119, 121, 124,
 126, 127, 129, 133, 135, 139, 140,
 143, 144, 148
- `assertComplex` (`checkComplex`), 32
- `assertCount`, 8, 145
- `assertCount` (`checkCount`), 36
- `assertDataFrame` (`checkDataFrame`), 39
- `assertDataTable` (`checkDataTable`), 43
- `assertDate` (`checkDate`), 47
- `assertDirectory` (`checkDirectoryExists`),
 51
- `assertDirectoryExists`
 (`checkDirectoryExists`), 51
- `assertDisjunct` (`checkDisjunct`), 53
- `assertDouble` (`checkDouble`), 54
- `assertEnvironment` (`checkEnvironment`), 59
- `assertFactor` (`checkFactor`), 60
- `assertFALSE` (`checkFALSE`), 65
- `assertFile` (`checkFileExists`), 66
- `assertFileExists` (`checkFileExists`), 66
- `assertFlag` (`checkFlag`), 68
- `assertFormula` (`checkFormula`), 70
- `assertFunction` (`checkFunction`), 71
- `assertInt`, 8, 145
- `assertInt` (`checkInt`), 73
- `assertInteger` (`checkInteger`), 76
- `assertIntegerish`, 8, 145
- `assertIntegerish` (`checkIntegerish`), 80

- assertList (checkList), 85
- assertLogical (checkLogical), 89
- assertMatrix (checkMatrix), 92
- assertMultiClass (checkMultiClass), 97
- assertNamed (checkNamed), 98
- assertNames (checkNames), 99
- assertNull (checkNull), 103
- assertNumber (checkNumber), 104
- assertNumeric (checkNumeric), 107
- assertOS (checkOS), 111
- assertPathForOutput
 - (checkPathForOutput), 112
- assertPermutation (checkPermutation), 114
- assertPOSIXct (checkPOSIXct), 116
- assertR6 (checkR6), 119
- assertRaw (checkRaw), 122
- assertScalar (checkScalar), 125
- assertScalarNA (checkScalarNA), 127
- assertSetEqual (checkSetEqual), 128
- assertString (checkString), 130
- assertSubset (checkSubset), 133
- assertTibble (checkTibble), 136
- assertTRUE (checkTRUE), 140
- assertVector (checkVector), 141

- check_access (checkAccess), 13
- check_array (checkArray), 14
- check_atomic (checkAtomic), 17
- check_atomic_vector
 - (checkAtomicVector), 20
- check_character (checkCharacter), 23
- check_choice (checkChoice), 28
- check_class (checkClass), 30
- check_complex (checkComplex), 32
- check_count (checkCount), 36
- check_data_frame (checkDataFrame), 39
- check_data_table (checkDataTable), 43
- check_date (checkDate), 47
- check_directory_exists
 - (checkDirectoryExists), 51
- check_disjunct (checkDisjunct), 53
- check_double (checkDouble), 54
- check_environment (checkEnvironment), 59
- check_factor (checkFactor), 60
- check_false (checkFALSE), 65
- check_file_exists (checkFileExists), 66
- check_flag (checkFlag), 68
- check_formula (checkFormula), 70
- check_function (checkFunction), 71
- check_int (checkInt), 73
- check_integer (checkInteger), 76
- check_integerish (checkIntegerish), 80
- check_list (checkList), 85
- check_logical (checkLogical), 89
- check_matrix (checkMatrix), 92
- check_multi_class (checkMultiClass), 97
- check_named (checkNamed), 98
- check_names (checkNames), 99
- check_null (checkNull), 103
- check_number (checkNumber), 104
- check_numeric (checkNumeric), 107
- check_os (checkOS), 111
- check_path_for_output
 - (checkPathForOutput), 112
- check_permutation (checkPermutation), 114
- check_posixct (checkPOSIXct), 116
- check_r6 (checkR6), 119
- check_raw (checkRaw), 122
- check_scalar (checkScalar), 125
- check_scalar_na (checkScalarNA), 127
- check_set_equal (checkSetEqual), 128
- check_string (checkString), 130
- check_subset (checkSubset), 133
- check_tibble (checkTibble), 136
- check_true (checkTRUE), 140
- check_vector (checkVector), 141
- checkAccess, 5, 13, 52, 68, 114
- checkArray, 5, 14, 20, 23, 28, 36, 43, 47, 51, 58, 60, 65, 71, 73, 80, 84, 88, 92, 96, 104, 111, 119, 125, 139, 143
- checkAtomic, 4, 17, 17, 23, 28, 36, 43, 51, 58, 60, 65, 71, 73, 80, 84, 88, 92, 96, 104, 111, 119, 125, 143, 144
- checkAtomicVector, 4, 17, 20, 20, 28, 36, 43, 51, 58, 60, 65, 71, 73, 80, 84, 88, 92, 96, 104, 111, 119, 125, 143, 144, 150
- checkCharacter, 4, 17, 20, 23, 23, 36, 43, 51, 58, 60, 65, 71, 73, 80, 84, 88, 92, 96, 104, 111, 119, 125, 130, 143
- checkChoice, 5, 28, 54, 115, 130, 135
- checkClass, 4, 30, 98, 99, 102, 122
- checkComplex, 4, 17, 20, 23, 28, 32, 43, 51, 58, 60, 65, 71, 73, 80, 84, 88, 92, 96, 104, 111, 119, 125, 143
- checkCount, 4, 36, 69, 76, 106, 126, 128, 133

- checkDataFrame, [5](#), [17](#), [20](#), [23](#), [28](#), [36](#), [39](#), [47](#),
[51](#), [58](#), [60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#),
[96](#), [104](#), [111](#), [119](#), [125](#), [139](#), [143](#)
- checkDataTable, [5](#), [17](#), [43](#), [43](#), [96](#), [139](#)
- checkDate, [5](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [47](#), [58](#),
[60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [111](#), [119](#), [125](#), [143](#)
- checkDirectory (checkDirectoryExists),
[51](#)
- checkDirectoryExists, [5](#), [14](#), [51](#), [68](#), [114](#)
- checkDisjunct, [5](#), [30](#), [53](#), [115](#), [130](#), [135](#)
- checkDouble, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#), [54](#),
[60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [107](#), [111](#), [119](#), [125](#), [143](#)
- checkEnvironment, [5](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#),
[51](#), [58](#), [59](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#),
[96](#), [104](#), [111](#), [119](#), [125](#), [143](#)
- checkFactor, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#), [58](#),
[60](#), [60](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [111](#), [119](#), [125](#), [143](#)
- checkFALSE, [65](#)
- checkFile (checkFileExists), [66](#)
- checkFileExists, [5](#), [14](#), [52](#), [66](#), [114](#)
- checkFlag, [4](#), [38](#), [68](#), [76](#), [106](#), [126](#), [128](#), [133](#)
- checkFormula, [5](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#),
[58](#), [60](#), [65](#), [70](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [111](#), [119](#), [125](#), [144](#)
- checkFunction, [5](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#),
[58](#), [60](#), [65](#), [71](#), [71](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [111](#), [119](#), [125](#), [144](#)
- checkInt, [4](#), [38](#), [69](#), [73](#), [106](#), [126](#), [128](#), [133](#)
- checkInteger, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#),
[58](#), [60](#), [65](#), [71](#), [73](#), [76](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [107](#), [111](#), [119](#), [125](#), [144](#)
- checkIntegerish, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#),
[51](#), [58](#), [60](#), [65](#), [71](#), [73](#), [80](#), [80](#), [88](#), [92](#),
[96](#), [104](#), [107](#), [111](#), [119](#), [125](#), [144](#), [150](#)
- checkList, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#), [58](#),
[60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [85](#), [92](#), [96](#),
[104](#), [111](#), [119](#), [125](#), [144](#)
- checkLogical, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#),
[58](#), [60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [89](#), [96](#),
[104](#), [111](#), [119](#), [125](#), [144](#)
- checkmate (checkmate-package), [4](#)
- checkmate-package, [4](#)
- checkMatrix, [5](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [47](#), [51](#),
[58](#), [60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [92](#),
[104](#), [111](#), [119](#), [125](#), [139](#), [144](#)
- checkMultiClass, [4](#), [32](#), [97](#), [99](#), [102](#), [122](#)
- checkNamed, [4](#), [10](#), [19](#), [22](#), [27](#), [32](#), [35](#), [42](#), [46](#),
[47](#), [57](#), [64](#), [79](#), [84](#), [87](#), [91](#), [95](#), [96](#), [98](#),
[98](#), [102](#), [110](#), [124](#), [139](#), [143](#)
- checkNames, [4](#), [32](#), [98](#), [99](#), [99](#)
- checkNull, [5](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#), [58](#),
[60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[103](#), [111](#), [119](#), [125](#), [144](#)
- checkNumber, [4](#), [38](#), [69](#), [76](#), [104](#), [126](#), [128](#), [133](#)
- checkNumeric, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#),
[58](#), [60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [107](#), [119](#), [125](#), [144](#)
- checkOS, [6](#), [111](#)
- checkPathForOutput, [5](#), [14](#), [52](#), [68](#), [112](#)
- checkPermutation, [5](#), [30](#), [54](#), [114](#), [130](#), [135](#)
- checkPOSIXct, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#),
[58](#), [60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [111](#), [116](#), [125](#), [144](#)
- checkR6, [5](#), [32](#), [98](#), [119](#)
- checkRaw, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#), [58](#), [60](#),
[65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#), [104](#),
[111](#), [119](#), [122](#), [144](#)
- checkScalar, [4](#), [38](#), [69](#), [76](#), [106](#), [125](#), [128](#), [133](#)
- checkScalarNA, [4](#), [38](#), [69](#), [76](#), [106](#), [126](#), [127](#),
[133](#)
- checkSetEqual, [5](#), [30](#), [54](#), [114](#), [115](#), [128](#), [135](#)
- checkString, [4](#), [23](#), [38](#), [69](#), [76](#), [106](#), [126](#), [128](#),
[130](#)
- checkSubset, [5](#), [10](#), [19](#), [27](#), [30](#), [35](#), [42](#), [46](#), [47](#),
[54](#), [57](#), [64](#), [79](#), [84](#), [87](#), [91](#), [95](#), [96](#),
[110](#), [115](#), [124](#), [130](#), [133](#), [139](#), [143](#)
- checkTibble, [5](#), [17](#), [43](#), [47](#), [96](#), [136](#)
- checkTRUE, [140](#)
- checkVector, [4](#), [17](#), [20](#), [23](#), [28](#), [36](#), [43](#), [51](#), [58](#),
[60](#), [65](#), [71](#), [73](#), [80](#), [84](#), [88](#), [92](#), [96](#),
[104](#), [111](#), [119](#), [125](#), [141](#)
- data.frame, [88](#)
- Date, [47](#), [50](#), [118](#)
- deparse, [153](#)
- expect_access (checkAccess), [13](#)
- expect_array (checkArray), [14](#)
- expect_atomic (checkAtomic), [17](#)
- expect_atomic_vector
(checkAtomicVector), [20](#)
- expect_character (checkCharacter), [23](#)
- expect_choice (checkChoice), [28](#)
- expect_class (checkClass), [30](#)

- `expect_complex` (`checkComplex`), 32
- `expect_count` (`checkCount`), 36
- `expect_data_frame` (`checkDataFrame`), 39
- `expect_data_table` (`checkDataTable`), 43
- `expect_date` (`checkDate`), 47
- `expect_directory`
 - (`checkDirectoryExists`), 51
- `expect_directory_exists`
 - (`checkDirectoryExists`), 51
- `expect_disjunct` (`checkDisjunct`), 53
- `expect_double` (`checkDouble`), 54
- `expect_environment` (`checkEnvironment`), 59
- `expect_factor` (`checkFactor`), 60
- `expect_file` (`checkFileExists`), 66
- `expect_file_exists` (`checkFileExists`), 66
- `expect_flag` (`checkFlag`), 68
- `expect_formula` (`checkFormula`), 70
- `expect_function` (`checkFunction`), 71
- `expect_int` (`checkInt`), 73
- `expect_integer` (`checkInteger`), 76
- `expect_integerish` (`checkIntegerish`), 80
- `expect_list` (`checkList`), 85
- `expect_logical` (`checkLogical`), 89
- `expect_matrix` (`checkMatrix`), 92
- `expect_multi_class` (`checkMultiClass`), 97
- `expect_names` (`checkNames`), 99
- `expect_number` (`checkNumber`), 104
- `expect_numeric` (`checkNumeric`), 107
- `expect_os` (`checkOS`), 111
- `expect_path_for_output`
 - (`checkPathForOutput`), 112
- `expect_permutation` (`checkPermutation`), 114
- `expect_posixct` (`checkPOSIXct`), 116
- `expect_r6` (`checkR6`), 119
- `expect_raw` (`checkRaw`), 122
- `expect_scalar` (`checkScalar`), 125
- `expect_scalar_na` (`checkScalarNA`), 127
- `expect_set_equal` (`checkSetEqual`), 128
- `expect_string` (`checkString`), 130
- `expect_subset` (`checkSubset`), 133
- `expect_that`, 13, 16, 19, 22, 27, 29, 31, 35, 38, 42, 47, 50, 52, 53, 58, 60, 64, 67, 69, 70, 72, 75, 79, 84, 88, 92, 96, 98, 102, 106, 110, 112, 113, 115, 119, 121, 124, 126, 127, 129, 133, 135, 139, 146, 149, 152
- `expect_tibble` (`checkTibble`), 136
- `expectation`, 14, 17, 20, 23, 28, 30, 32, 36, 38, 42, 47, 50, 52, 54, 58, 60, 65–67, 69, 71, 73, 75, 80, 84, 88, 92, 96, 98, 99, 102, 103, 106, 111, 112, 114, 115, 119, 122, 124, 126, 128, 129, 133, 135, 139, 140, 143, 145, 150
- `fmatch`, 29, 53, 129, 134, 135
- `grepl`, 27, 132
- `inherits`, 31, 41, 46, 87, 97, 121, 138
- `is.atomic`, 6–8, 17, 20
- `is.list`, 88
- `is.numeric`, 107
- `is.vector`, 20, 143
- `isTRUE`, 147
- `makeAssertCollection`
 - (`AssertCollection`), 12
- `makeAssertion`, 144, 146, 148
- `makeAssertionFunction` (`makeAssertion`), 144
- `makeExpectation`, 145, 145, 148
- `makeExpectationFunction`
 - (`makeExpectation`), 145
- `makeTest`, 145, 146, 147
- `makeTestFunction` (`makeTest`), 147
- `match`, 29, 53, 129, 135
- `match.arg`, 148
- `matchArg`, 148
- `Ops.Date`, 50
- `pairlist`, 88
- `parent.frame`, 145–147
- `POSIXct`, 116
- `qassert`, 5, 149, 151, 152
- `qassertr`, 5, 151, 151
- `qexpect`, 151
- `qexpect` (`qassert`), 149
- `qexpectr` (`qassertr`), 151
- `qtest`, 151, 152
- `qtest` (`qassert`), 149
- `qtestr`, 151
- `qtestr` (`qassertr`), 151
- `register_test_backend`, 152

- reportAssertions (AssertCollection), 12
- substitute, 153
- test_access (checkAccess), 13
- test_array (checkArray), 14
- test_atomic (checkAtomic), 17
- test_atomic_vector (checkAtomicVector), 20
- test_character (checkCharacter), 23
- test_choice (checkChoice), 28
- test_class (checkClass), 30
- test_complex (checkComplex), 32
- test_count (checkCount), 36
- test_data_frame (checkDataFrame), 39
- test_data_table (checkDataTable), 43
- test_date (checkDate), 47
- test_directory (checkDirectoryExists), 51
- test_directory_exists (checkDirectoryExists), 51
- test_disjunct (checkDisjunct), 53
- test_double (checkDouble), 54
- test_environment (checkEnvironment), 59
- test_factor (checkFactor), 60
- test_false (checkFALSE), 65
- test_file_exists (checkFileExists), 66
- test_flag (checkFlag), 68
- test_formula (checkFormula), 70
- test_function (checkFunction), 71
- test_int (checkInt), 73
- test_integer (checkInteger), 76
- test_integerish (checkIntegerish), 80
- test_list (checkList), 85
- test_logical (checkLogical), 89
- test_matrix (checkMatrix), 92
- test_multi_class (checkMultiClass), 97
- test_named (checkNamed), 98
- test_names (checkNames), 99
- test_null (checkNull), 103
- test_number (checkNumber), 104
- test_numeric (checkNumeric), 107
- test_os (checkOS), 111
- test_path_for_output (checkPathForOutput), 112
- test_permutation (checkPermutation), 114
- test_posixct (checkPOSIXct), 116
- test_r6 (checkR6), 119
- test_raw (checkRaw), 122
- test_scalar (checkScalar), 125
- test_scalar_na (checkScalarNA), 127
- test_set_equal (checkSetEqual), 128
- test_string (checkString), 130
- test_subset (checkSubset), 133
- test_tibble (checkTibble), 136
- test_true (checkTRUE), 140
- test_vector (checkVector), 141
- testAccess (checkAccess), 13
- testArray (checkArray), 14
- testAtomic (checkAtomic), 17
- testAtomicVector (checkAtomicVector), 20
- testCharacter (checkCharacter), 23
- testChoice (checkChoice), 28
- testClass (checkClass), 30
- testComplex (checkComplex), 32
- testCount (checkCount), 36
- testDataFrame (checkDataFrame), 39
- testDataTable (checkDataTable), 43
- testDate (checkDate), 47
- testDirectory (checkDirectoryExists), 51
- testDirectoryExists (checkDirectoryExists), 51
- testDisjunct (checkDisjunct), 53
- testDouble (checkDouble), 54
- testEnvironment (checkEnvironment), 59
- testFactor (checkFactor), 60
- testFALSE (checkFALSE), 65
- testFile (checkFileExists), 66
- testFileExists (checkFileExists), 66
- testFlag (checkFlag), 68
- testFormula (checkFormula), 70
- testFunction (checkFunction), 71
- testInt (checkInt), 73
- testInteger (checkInteger), 76
- testIntegerish (checkIntegerish), 80
- testList (checkList), 85
- testLogical (checkLogical), 89
- testMatrix (checkMatrix), 92
- testMultiClass (checkMultiClass), 97
- testNamed (checkNamed), 98
- testNames (checkNames), 99
- testNull (checkNull), 103
- testNumber (checkNumber), 104
- testNumeric (checkNumeric), 107
- testOS (checkOS), 111
- testPathForOutput (checkPathForOutput), 112

testPermutation (checkPermutation), 114
testPOSIXct (checkPOSIXct), 116
testR6 (checkR6), 119
testRaw (checkRaw), 122
testScalar (checkScalar), 125
testScalarNA (checkScalarNA), 127
testSetEqual (checkSetEqual), 128
testString (checkString), 130
testSubset (checkSubset), 133
testthat, 150
testTibble (checkTibble), 136
testTRUE (checkTRUE), 140
testVector (checkVector), 141
typeof, 30, 54, 115, 130, 135

unique, 88

vname, 10, 11, 13, 16, 19, 22, 23, 27, 29, 31,
35, 38, 42, 47, 50, 52, 53, 58, 60, 64,
65, 67, 69, 70, 72, 75, 79, 84, 88, 91,
92, 96–99, 102, 103, 106, 110, 112,
113, 115, 119, 121, 122, 124, 126,
127, 129, 133, 135, 139, 140, 143,
148, 149, 152, 153

wf, 6, 153
wl (wf), 153