Package 'NVIcheckmate'

December 16, 2024

Title Extension of checkmate with argument checking adapted for NVIverse

Version 0.8.0.9000

Date 2024-##-##

Description Extends `checkmate` with functions for argument checking that are adapted for NVIverse. NVIcheckmate is intended to be used together with `checkmate`.

URL https://github.com/NorwegianVeterinaryInstitute/NVIcheckmate

BugReports https://github.com/NorwegianVeterinaryInstitute/NVIcheckmate/issues

License BSD_3_clause + file LICENSE

Encoding UTF-8

Imports checkmate (>= 2.1.0),
DBI,
keyring

Roxygen list(markdown = FALSE)

RoxygenNote 7.3.1

Suggests covr,
desc,
devtools,
findInFiles,
knitr,
R.rsp,
remotes,
rmarkdown,
roxygen2,
RPostgreSQL,
RODBC,
rstudioapi,
testthat,
NVIpackager,
NVIrpackages

Remotes NorwegianVeterinaryInstitute/NVIpackager,
NorwegianVeterinaryInstitute/NVIrpackages,

Collate 'NVIcheckmate-package.R'
'assert.R'
'makeAssertionFunction.R'
'assert_character.R'

'assert_data_frame.R'
'assert_disjunct.R'
'assert_integer.R'
'assert_integerish.R'
'assert_names.R'
'check_choice_character.R'
'check_credentials.R'
'check_non_null.R'
'check_odbc_channel.R'
'check_package.R'
'check_subset_character.R'
'helper.R'
'match_arg.R'

VignetteBuilder knitr, R.rsp

Contents

`assert` *Combine multiple checks into one assertion*

Description

You can call this function with an arbitrary number of of `check*` functions, i.e. functions provided by the packages `checkmate`, `NVIcheckmate` or your own functions which return `TRUE` on success and the error message as `character(1)` otherwise.

Usage

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

Arguments

...	[any] List of calls to check functions.
combine	[character(1)] "or" or "and" to combine the check functions with an OR or AND, respectively.
.var.name	[character(1)] Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

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.

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

---

assert_character *Check if an argument is a vector of type character*

Description

Performs assertions if an argument is a vector of type character.

Usage

```
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,
  null.ok = FALSE,
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

x	[any] Object to check.
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(1L)] Regular expression as used in grepl. All non-missing elements of x must comply to this pattern.
fixed	[character(1)] Substring to detect in x. Will be used as pattern in grepl with option fixed set to TRUE. All non-missing elements of x must contain this substring.
ignore.case	[logical(1)] See grepl. Default is FALSE.
any.missing	[logical(1)] Are vectors with missing values allowed? Default is TRUE.
all.missing	[logical(1)] Are vectors with 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.

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 error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

The assertions are based on checkmate::checkCharacter. NVIcheckmate::assert_character differs from checkmate::assert_character in including the argument comment = . The help is updated to reflect the changes.

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

Value

If the check is successful, the function assert_character return x invisibly.
If the check is not successful, assert_character throws an error message.

---

assert_data_frame *Check if an argument is vector of type data frame*

Description

Check if an argument is vector of type data frame.

Usage

```
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 = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

x	[any] Object to check.
types	[character] Character vector of class names. Each list element must inherit from at least one of the provided types. The types "logical", "integer", "integerish", "double", "numeric", "complex", "character", "factor", "atomic", "vector" "atomicvector", "array", "matrix", "list", "function", "environment" and "null" are supported. For other types inherits is used as a fallback to check x's inheritance. Defaults to character(0) (no check).
any.missing	[logical(1)] Are vectors with missing values allowed? Default is TRUE.
all.missing	[logical(1)] Are vectors with no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
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 checkmate::check_named for possible values. Note that you can use checkmate::check_subset to check for a specific set of names.
col.names	[character(1)] Check for column names. Default is "NULL" (no check). See checkmate::check_named for possible values. Note that you can use checkmate::check_subset 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 error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

The assertions are based on checkmate::check_data_frame. NVIcheckmate::assert_data_frame differs from checkmate::assert_data_frame in including the argument comment. The help is updated to reflect the changes.

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 function assert_Integer return x invisibly, whereas check_Integer return TRUE.
If the check is not successful, assert_Integer throws an error message and check_Integer returns a string with the error message.

If the check is successful, the function assert_data_frame return x invisibly.
If the check is not successful, assert_data_frame throws an error message.

---

assert_disjunct *Check if an argument is disjunct from a given set*

Description

Check if an argument is disjunct from a given set.

Usage

```
assert_disjunct(
  x,
  y,
  fmatch = FALSE,
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

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. fastmatch::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 error messages. Defaults to the heuristic implemented in vname.

comment [character(1)]
Extra information to be appended to the standard error message in assertions.

add [AssertCollection]
Collection to store assertions. See AssertCollection.

Details

The assertions are based on checkmate::checkDisjunct. NVIcheckmate::assert_disjunct differs from checkmate::assert_disjunct in including the argument comment = . The help is updated to reflect the changes.

Value

If the check is successful, the function assert_disjunct return x invisibly.
If the check is not successful, assert_disjunct throws an error message.

---

assert_integer *Check if an argument is vector of type integer*

Description

Check if an argument is vector of type integer.

Usage

```
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 = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

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 error messages. Defaults to the heuristic implemented in vname.

comment [character(1)]
Extra information to be appended to the standard error message in assertions.

add [AssertCollection]
Collection to store assertions. See AssertCollection.

Details

The assertions are based on checkmate::checkInteger. NVIcheckmate::assert_integer differs from checkmate::assert_integer in including the argument comment = . The help is updated to reflect the changes.

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 function assert_Integer return x invisibly, whereas check_Integer return TRUE.

If the check is not successful, `assert_Integer` throws an error message and `check_Integer` returns a string with the error message.

If the check is successful, the function `assert_integer` return `x` invisibly.
If the check is not successful, `assert_integer` throws an error message.

`assert_integerish` *Check if an argument is vector of type integer*

Description

Check if an argument is vector of type integer.

Usage

```
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,
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

`x`	[any] Object to check.
`tol`	[double(1)] Numerical tolerance used to check whether a double or complex can be converted. Default is `sqrt(.Machine$double.eps)`.
`lower`	[numeric(1)] Lower value all elements of `x` must be greater than or equal to.
`upper`	[numeric(1)] Upper value all elements of `x` must be lower than or equal to.
`any.missing`	[logical(1)] Are vectors with missing values allowed? Default is `TRUE`.
`all.missing`	[logical(1)] Are vectors with 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 error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

The assertions are based on checkmate::checkIntegerish. NVIcheckmate::assert_integerish differs from checkmate::assert_integerish in including the argument comment = . The help is updated to reflect the changes.

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 function assert_Integerish return x invisibly, whereas check_Integerish return TRUE.
If the check is not successful, assert_Integerish throws an error message and check_Integerish returns a string with the error message.

If the check is successful, the function assert_integerish return x invisibly.
If the check is not successful, assert_integerish throws an error message.

`assert_names` *Check names to comply to specific rules*

Description

Performs assertions with various checks on character vectors, usually names.

Usage

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

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

identical.to	[character] Names provided in x must be identical to the vector identical.to. Use this argument instead of permutation.of if the order of the names is relevant.
disjunct.from	[character] Names provided in x must may not be present in the vector disjunct.from.
what	[character(1)] Type of name vector to check, e.g. "names" (default), "colnames" or "rownames".
.var.name	[character(1)] Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

The assertions are based on checkmate::checkNames. NVIcheckmate::assert_names differs from checkmate::assert_names in including the argument comment = . The help is updated to reflect the changes.

Value

If the check is successful, the function assert_names return x invisibly.
If the check is not successful, assert_names throws an error message.

---

check_choice_character

Check if an object is an element of a given set

Description

Check if an object is an element of a given set in the object name. The function is based on checkmate::check_choice, but includes the argument ignore.case.

Usage

```
check_choice_character(
  x,
  choices,
  null.ok = FALSE,
  ignore.case = FALSE,
  fmatch = FALSE
)

assert_choice_character(
  x,
  choices,
  null.ok = FALSE,
```

```
  ignore.case = FALSE,
  fmatch = FALSE,
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

x	[any] Object to check.
choices	[character] 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.
ignore.case	[logical(1)] Case is ignored if TRUE. 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. fastmatch::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 error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

The object must be of type character. The check is intended for functions were using camelCase may make the argument easier to remember. Therefore, the function will ignore case if ignore.case = TRUE.

Value

Depending on the function prefix:
If the check is successful, the function assert_choice_character return x invisibly, whereas check_choice_character return TRUE.
If the check is not successful, assert_choice_character throws an error message and check_choice_character returns a string with the error message.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# returns TRUE
check_choice_character(x = "APPLE",
                       choices = c("Apple", "Pear", "Orange"),
                       null.ok = FALSE,
                       ignore.case = TRUE,
                       fmatch = FALSE)
```

---

check_credentials *Check if an object is a service for which credentials are saved in the user profile*

Description

Check if credentials are saved in the user profile for the service.

Usage

```
check_credentials(x)

assert_credentials(
  x,
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

x	[any] Object to check.
.var.name	[character(1)] Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Value

Depending on the function prefix:
If the check is successful, the function assert_credentials return x invisibly, whereas check_credentials return TRUE.
If the check is not successful, assert_credentials throws an error message and check_credentials returns a string with the error message.

Examples

```
check_credentials(x = "PJS")
```

check_non_null *Check if two or more arguments are NULL or NA*

Description

Check whether two or more arguments that may replace each other are NULL or NA.

Usage

```
check_non_null(x)

assert_non_null(x, .var.name = checkmate::vname(x), comment = NULL, add = NULL)
```

Arguments

x	[any] List with objects to check.
.var.name	[character(1)] Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

In some functions, one argument may replace another. The check is used to ensure that at least on of the arguments have a value. Specific checks for each argument should be used in addition.

Value

Depending on the function prefix:
If the check is successful, the function assert_non_null return x invisibly, whereas check_non_null return TRUE.
If the check is not successful, assert_non_null throws an error message and check_non_null returns a string with the error message.

Examples

```
data <- NULL
nrows_in_data <- 56
check_non_null(x = list(data, nrows_in_data))
```

check_odbc_channel *Check if an object is an open ODBC-channel*

Description

Check if an object is an open ODBC-channel.

Usage

```
check_odbc_channel(x, dbservice = NULL, dbinterface = "odbc")

assert_odbc_channel(
  x,
  dbservice = NULL,
  dbinterface = "odbc",
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

x	[any] Object to check.
dbservice	[character(1)] The database the channel should be connect to.
dbinterface	[character(1)] The R-package that is used for interface towards the data base. Defaults to NULL.
.var.name	[character(1)] Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

The check uses RODBC:::odbcValidChannel to check if the argument is an open ODBC-channel. Be aware that this function is an internal RODBC-function and may change within that package without warning.

Value

Depending on the function prefix:
If the check is successful, the function assert_odbc_channel return x invisibly, whereas check_odbc_channel return TRUE.
If the check is not successful, assert_odbc_channel throws an error message and check_odbc_channel returns a string with the error message.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# login to PJS
journal_rapp <- login("PJS")

# error check:
check_odbc_channel(x = "journal_rapp", dbservice = "PJS")

## End(Not run)
```

---

check_package *Check if an object is an installed or attached package*

Description

Check if a package is installed and if it is installed, check if the version is equal to or higher than the required version or check if a package is attached.

Usage

```
check_package(x, version = NULL, type = "installed")

assert_package(
  x,
  version = NULL,
  type = "installed",
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

x	[any] Object to check.
version	[character(1)] The required version of the installed package. May be NULL.
type	[character(1)] Type of formal check(s) to perform on the package. **installed:** Checks if x is installed. **attached:** Checks if x is attached.
.var.name	[character(1)] Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.

comment [character(1)]
Extra information to be appended to the standard error message in assertions.

add [AssertCollection]
Collection to store assertions. See AssertCollection.

Value

Depending on the function prefix:
If the check is successful, the function assert_package return x invisibly, whereas check_package return TRUE.
If the check is not successful, assert_package throws an error message and check_package returns a string with the error message.

Examples

```
# returns TRUE i.e. no error message
check_package(x = "checkmate", version = "2.0.0", type = "installed")

# returns an error message
check_package(x = "nopackage", type = "installed")
check_package(x = "nopackage", type = "attached")
```

---

check_subset_character

Check if an argument is a subset of a given set

Description

Check if an object is a subset of a given set in the object name. The function is based on checkmate::check_subset, but includes the argument ignore.case.

Usage

```
check_subset_character(
  x,
  choices,
  ignore.case = FALSE,
  empty.ok = TRUE,
  fmatch = FALSE
)

assert_subset_character(
  x,
  choices,
  ignore.case = FALSE,
  empty.ok = TRUE,
  fmatch = FALSE,
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

x	[any] Object to check.
choices	[character] Set of possible values. May be empty.
ignore.case	[logical(1)] Case is ignored if TRUE. Default is FALSE.
empty.ok	[logical(1)] Treat zero-length x as subset of any set choices (this includes NULL)? Default is TRUE.
fmatch	[logical(1)] Use the set operations implemented in fmatch in package **fastmatch**. If **fastmatch** is not installed, this silently falls back to match. fastmatch::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 error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

The object must be of type character. The check is intended for functions were using camelCase may make the argument easier to remember. Therefore, the function will ignore case if ignore.case = TRUE.

Value

Depending on the function prefix:
If the check is successful, the function assert_subset_character return x invisibly, whereas check_subset_character return TRUE.
If the check is not successful, assert_subset_character throws an error message and check_subset_character returns a string with the error message.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# returns TRUE
check_subset_character(x = "Apple",
                       choices = c("apple", "pear", "orange", "banana"),
                       ignore.case = TRUE)
check_subset_character(x = c("Apple", "Pear"),
                       choices = c("apple", "pear", "orange", "banana"),
                       ignore.case = TRUE)
```

```
# returns a message
check_subset_character(x = "Tomato",
                       choices = c("apple", "pear", "orange", "banana"),
                       ignore.case = TRUE)
```

`makeAssertionFunction` *Turn a Check into an Assertion*

Description

`makeAssertionFunction` can be used to automatically create an assertion function based on a check function (see example). This is a modification of `checkmate::makeAssertionFunction` that includes the argument comment in the assertion function.

Usage

```
makeAssertionFunction(
  check.fun,
  c.fun = NULL,
  coerce = FALSE,
  env = parent.frame()
)
```

Arguments

`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 `CallExternal`. Useful if `check.fun` is just a simple wrapper.
`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`, see `sys.parent`.

Details

The code is imported from `checkmate`. The modifications in the code is marked. The argument use.namespace is deleted as `checkmate::makeAssertion` and `checkmate::vname` always should be used.

match_arg *Argument Verification Using Partial Matching*

Description

match.arg matches arguments against a table of candidate values as specified by choices. Matching is done using grep, and arguments may be abbreviated. If ignore.case = TRUE, case is ignored when performing the matching.

Usage

```
match_arg(
  x,
  choices,
  several.ok = FALSE,
  ignore.case = FALSE,
  .var.name = checkmate::vname(x),
  comment = NULL,
  add = NULL
)
```

Arguments

x	[character] User provided argument to match.
choices	[character] Candidates to match x with.
several.ok	[logical(1)] If TRUE, x should be allowed to have more than one element.
ignore.case	[logical(1)] Case is ignored if TRUE. Default is FALSE.
.var.name	[character(1)] Name of the checked object to print in error messages. Defaults to the heuristic implemented in vname.
comment	[character(1)] Extra information to be appended to the standard error message in assertions.
add	[AssertCollection] Collection to store assertions. See AssertCollection.

Details

Partial Argument Matching

This is an extensions to matchArg with support for ignore.case and comment. matchArg is an extension of match.arg with support for AssertCollection. The behavior is very similar to match.arg, with a few exceptions: * NULL is not a valid value for x. * When several.ok = TRUE, it is required that all values for x match a value in choices and that each value in x matches only one value in choices. * When several.ok = FALSE, it is required that length(x) == 1 and that x matches one and only one value in choices.

Value

Character vector with subset of `choices`.

Examples

```
match_arg("a", choices = c("Apple", "Banana"), ignore.case = TRUE)
match_arg(c("a", "Ban"), choices = c("Apple", "Banana"), several.ok = TRUE, ignore.case = TRUE)
```

Index