Package ‘admiraldev’

December 15, 2023

Type Package

Title Utility Functions and Development Tools for the Admiral Package Family

Version 1.0.0

Description Utility functions to check data, variables and conditions for functions used in 'admiral' and 'admiral' extension packages. Additional utility helper functions to assist developers with maintaining documentation, testing and general upkeep of 'admiral' and 'admiral' extension packages.

License Apache License (>= 2)

BugReports https://github.com/pharmaverse/admiraldev/issues

URL https://pharmaverse.github.io/admiraldev/,
https://github.com/pharmaverse/admiraldev/

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

Depends R (>= 3.5)

Imports dplyr (>= 1.0.5), hms (>= 0.5.3), lifecycle (>= 0.1.0), lubridate (>= 1.7.4), magrittr (>= 1.5), purrr (>= 0.3.3), rlang (>= 0.4.4), stringr (>= 1.4.0), tidyr (>= 1.0.2), tidyselect (>= 1.0.0)

Suggests pharmaversesdtm, devtools, diffdf, lintr, pkgdown, testthat (>= 3.0.0), knitr, methods, miniUI, rmarkdown, roxygen2, rstudioapi, spelling, styler, tibble, usethis, covr, DT, htmltools

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Ben Straub [aut, cre],
Stefan Bundfuss [aut],
Jeffrey Dickinson [aut],

Ross Farrugia [aut],
Pooja Kumari [aut],
Edoardo Mancini [aut],
Sadchla Mascary [aut],
Zelos Zhu [aut],
Ania Golab [ctb],
Samia Kabi [ctb],
Syed Mubasheer [ctb],
Thomas Neitmann [ctb],
Ondrej Slama [ctb],
F. Hoffmann-La Roche AG [cph, fnd],
GlaxoSmithKline LLC [cph, fnd]

Maintainer Ben Straub <ben.x.straub@gsk.com>

Repository CRAN

Date/Publication 2023-12-15 13:20:06 UTC

R topics documented:

`add_suffix_to_vars` *Add a Suffix to Variables in a List of Expressions*

Description

Add a suffix to variables in a list of expressions

Usage

```
add_suffix_to_vars(order, vars, suffix)
```

Arguments

`order`	List of expressions *Permitted Values*: list of variables or `desc(<variable>)` function calls created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))`
`vars`	Variables to change *Permitted Values*: list of variables created by `exprs()`
`suffix`	Suffix *Permitted Values*: A character scalar

Value

The list of expression where for each element the suffix (`suffix`) is added to every symbol specified for `vars`

See Also

Helpers for working with Quosures: `expr_c()`, `replace_symbol_in_expr()`, `replace_values_by_names()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(rlang)

add_suffix_to_vars(exprs(ADT, desc(AVAL), AVALC), vars = exprs(AVAL), suffix = ".join")
```

`anti_join` *Join Functions*

Description

The `*_join()` functions from `{dplyr}` without a warning on different attributes in datasets.

Usage

```
anti_join(x, y, by = NULL, copy = FALSE, ...)

inner_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

left_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)
```

Arguments

`x`	`data.frame`
`y`	`data.frame`
`by`	`character` vector
`copy`	`logical`
`...`	Additional arguments
`suffix`	`character` vector

Value

`data.frame`

`arg_name` *Extract Argument Name from an Expression*

Description

Extract Argument Name from an Expression

Usage

```
arg_name(expr)
```

Arguments

`expr`	An expression created inside a function using `substitute()`

Value

`character` vector

See Also

Developer Utility Functions: `%notin%()`, `%or%()`, `contains_vars()`, `convert_dtm_to_dtc()`, `extract_vars()`, `filter_if()`, `friendly_type_of()`, `valid_time_units()`, `vars2chr()`

assert_atomic_vector *Is an Argument an Atomic Vector?*

Description

Checks if an argument is an atomic vector

Usage

```
assert_atomic_vector(arg, optional = FALSE)
```

Arguments

arg	A function argument to be checked
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown

Value

The function throws an error if arg is not an atomic vector. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: assert_character_scalar(), assert_character_vector(), assert_data_frame(), assert_date_vector(), assert_expr_list(), assert_expr(), assert_filter_cond(), assert_function(), assert_integer_scalar(), assert_list_element(), assert_list_of(), assert_logical_scalar(), assert_named(), assert_numeric_vector(), assert_one_to_one(), assert_param_does_not_exist(), assert_s3_class(), assert_same_type(), assert_symbol(), assert_unit(), assert_vars(), assert_varval_list()

Examples

```
example_fun <- function(x) {
  assert_atomic_vector(x)
}

example_fun(1:10)

try(example_fun(list(1, 2)))
```

assert_character_scalar

Is an Argument a Character Scalar (String)?

Description

Checks if an argument is a character scalar and (optionally) whether it matches one of the provided `values`.

Usage

```
assert_character_scalar(
  arg,
  values = NULL,
  case_sensitive = TRUE,
  optional = FALSE
)
```

Arguments

`arg`	A function argument to be checked
`values`	A `character` vector of valid values for `arg`. Values is converted to a lower case vector if case_sensitive = FALSE is used.
`case_sensitive`	Should the argument be handled case-sensitive? If set to `FALSE`, the argument is converted to lower case for checking the permitted values and returning the argument.
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a character vector or if `arg` is a character vector but of length > 1 or if its value is not one of the `values` specified. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(msg_type) {
  assert_character_scalar(msg_type, values = c("warning", "error"))
}

example_fun("warning")

try(example_fun("message"))

try(example_fun(TRUE))

# handling arguments case-insensitive
example_fun2 <- function(msg_type) {
  msg_type <- assert_character_scalar(
    msg_type,
    values = c("warning", "error"),
    case_sensitive = FALSE
  )
  if (msg_type == "warning") {
    print("A warning was requested.")
  }
}

example_fun2("Warning")
```

---

assert_character_vector

Is an Argument a Character Vector?

Description

Checks if an argument is a character vector

Usage

```
assert_character_vector(arg, values = NULL, named = FALSE, optional = FALSE)
```

Arguments

`arg`	A function argument to be checked
`values`	A `character` vector of valid values for `arg`
`named`	If set to `TRUE`, an error is issued if not all elements of the vector are named.
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a character vector or if any element is not included in the list of valid values. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(chr) {
  assert_character_vector(chr)
}

example_fun(letters)

try(example_fun(1:10))

example_fun2 <- function(chr) {
  assert_character_vector(chr, named = TRUE)
}

try(example_fun2(c(alpha = "a", "b", gamma = "c")))
```

`assert_data_frame` *Is an Argument a Data Frame?*

Description

Checks if an argument is a data frame and (optionally) whether is contains a set of required variables

Usage

```
assert_data_frame(
  arg,
  required_vars = NULL,
  check_is_grouped = TRUE,
  optional = FALSE
)
```

Arguments

`arg`	A function argument to be checked
`required_vars`	A list of variables created using `exprs()`
`check_is_grouped`	Throw an error is `dataset` is grouped? Defaults to `TRUE`.
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a data frame or if `arg` is a data frame but misses any variable specified in `required_vars`. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
library(pharmaversesdtm)
library(dplyr, warn.conflicts = FALSE)
library(rlang)
data(dm)

example_fun <- function(dataset) {
  assert_data_frame(dataset, required_vars = exprs(STUDYID, USUBJID))
}

example_fun(dm)

try(example_fun(select(dm, -STUDYID)))

try(example_fun("Not a dataset"))
```

`assert_date_var` *Is a Variable in a Dataset a Date or Datetime Variable?*

Description

Checks if a variable in a dataset is a date or datetime variable

Usage

```
assert_date_var(dataset, var, dataset_name = NULL, var_name = NULL)
```

Arguments

`dataset`	The dataset where the variable is expected
`var`	The variable to check
`dataset_name`	The name of the dataset. If the argument is specified, the specified name is displayed in the error message.
`var_name`	The name of the variable. If the argument is specified, the specified name is displayed in the error message.

Value

The function throws an error if `var` is not a date or datetime variable in `dataset` and returns the input invisibly otherwise.

Examples

```
library(tibble)
library(lubridate)
library(rlang)

example_fun <- function(dataset, var) {
  var <- assert_symbol(enexpr(var))
  assert_date_var(dataset = dataset, var = !!var)
}

my_data <- tribble(
  ~USUBJID, ~ADT,
  "1",      ymd("2020-12-06"),
  "2",      ymd("")
)

example_fun(
  dataset = my_data,
  var = ADT
)

try(example_fun(
  dataset = my_data,
  var = USUBJID
))

example_fun2 <- function(dataset, var) {
  var <- assert_symbol(enexpr(var))
  assert_date_var(
    dataset = dataset,
    var = !!var,
    dataset_name = "your_data",
    var_name = "your_var"
  )
}

try(example_fun2(
  dataset = my_data,
  var = USUBJID
))
```

`assert_date_vector` *Is an object a date or datetime vector?*

Description

Check if an object/vector is a date or datetime variable without needing a dataset as input

Usage

```
assert_date_vector(arg, optional = FALSE)
```

Arguments

`arg`	The function argument to be checked
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then the function `assert_date_vector` exits early and throw and error.

Value

The function returns an error if `arg` is missing, or not a date or datetime variable but otherwise returns an invisible output.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(arg) {
  assert_date_vector(arg)
}

example_fun(
  as.Date("2022-01-30", tz = "UTC")
)
try(example_fun("1993-07-14"))
```

`assert_expr` *Assert Argument is an Expression*

Description

Assert Argument is an Expression

Usage

```
assert_expr(arg, optional = FALSE)
```

Arguments

`arg`	A function argument to be checked
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not an expression, i.e. either a symbol or a call, or returns the input invisibly otherwise

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

`assert_expr_list` *Is an Argument a List of Expressions?*

Description

Checks if the argument is a list of expressions.

Usage

```
assert_expr_list(
  arg,
  required_elements = NULL,
  named = FALSE,
  optional = FALSE
)
```

Arguments

`arg`	A function argument to be checked
`required_elements`	A `character` vector of names that must be present in `arg`
`named`	If set to `TRUE`, an error is issued if not all elements of the list are named.
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown.

Value

The function throws an error if `arg` is not a list of expressions. Otherwise, the input it returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
library(rlang)

example_fun <- function(vars) {
  assert_expr_list(vars)
}
example_fun(exprs(DTHDOM = "AE", DTHSEQ = AESEQ))

try(example_fun(exprs("AE", DTSEQ = AESEQ, !!list("a"))))
```

`assert_filter_cond` *Is an Argument a Filter Condition?*

Description

Is an Argument a Filter Condition?

Usage

```
assert_filter_cond(arg, optional = FALSE)
```

Arguments

`arg`	Quosure - filtering condition.
`optional`	Logical - is the argument optional? Defaults to `FALSE`.

Details

Check if `arg` is a suitable filtering condition to be used in functions like `subset` or `dplyr::filter`.

Value

Performs necessary checks and returns `arg` if all pass. Otherwise throws an informative error.

See Also

Checks for valid input and returns warning or errors messages: assert_atomic_vector(), assert_character_scalar(), assert_character_vector(), assert_data_frame(), assert_date_vector(), assert_expr_list(), assert_expr(), assert_function(), assert_integer_scalar(), assert_list_element(), assert_list_of(), assert_logical_scalar(), assert_named(), assert_numeric_vector(), assert_one_to_one(), assert_param_does_not_exist(), assert_s3_class(), assert_same_type(), assert_symbol(), assert_unit(), assert_vars(), assert_varval_list()

Examples

```
library(pharmaversesdtm)
library(dplyr, warn.conflicts = FALSE)
library(rlang)
data(dm)

# typical usage in a function as an argument check
example_fun <- function(dat, x) {
  x <- assert_filter_cond(enquo(x))
  filter(dat, !!x)
}

example_fun(dm, AGE == 64)

try(example_fun(dm, USUBJID))
```

---

assert_function *Is Argument a Function?*

Description

Checks if the argument is a function and if all expected arguments are provided by the function.

Usage

```
assert_function(arg, params = NULL, optional = FALSE)
```

Arguments

arg	A function The function to be checked
params	A character vector A character vector of expected argument names for the aforementioned function in arg. If ellipsis, ..., is included in the function formals of the function in arg, this argument, params will be ignored, accepting all values of the character vector.
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown.

Value

The function throws an error

- if the argument is not a function or
- if the function does not provide all arguments as specified for the `params` argument (assuming ellipsis is not in function formals)

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(fun) {
  assert_function(fun, params = c("x"))
}

example_fun(mean)

try(example_fun(1))

try(example_fun(sum))
```

`assert_function_param` *Assert Argument is a Parameter of a Function*

Description

[Deprecated]

This function is *deprecated*, please use `assert_function()` instead.

Usage

```
assert_function_param(arg, params)
```

Arguments

`arg`	The name of a function passed as a string
`params`	A character vector of function parameters

Value

The function throws an error if any elements of `params` is not an argument of the function given by `arg`

See Also

Other deprecated: `assert_has_variables()`, `assert_named_exprs()`

`assert_has_variables` *Does a Dataset Contain All Required Variables?*

Description

[Deprecated]

This function is *deprecated*, please use `assert_data_frame()` instead.

Usage

```
assert_has_variables(dataset, required_vars)
```

Arguments

`dataset`	A `data.frame`
`required_vars`	A `character` vector of variable names

Details

Checks if a dataset contains all required variables

Value

The function throws an error if any of the required variables are missing in the input dataset. Otherwise, the dataset is returned invisibly.

See Also

Other deprecated: `assert_function_param()`, `assert_named_exprs()`

assert_integer_scalar *Is an Argument an Integer Scalar?*

Description

Checks if an argument is an integer scalar

Usage

```
assert_integer_scalar(arg, subset = "none", optional = FALSE)
```

Arguments

`arg`	A function argument to be checked
`subset`	A subset of integers that `arg` should be part of. Should be one of `"none"` (the default), `"positive"`, `"non-negative"` or `"negative"`.
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not an integer belonging to the specified `subset`. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(num1, num2) {
  assert_integer_scalar(num1, subset = "positive")
  assert_integer_scalar(num2, subset = "negative")
}

example_fun(1, -9)

try(example_fun(1.5, -9))

try(example_fun(2, 0))

try(example_fun("2", 0))
```

`assert_list_element` *Is an Element of a List of Lists/Classes Fulfilling a Condition?*

Description

Checks if the elements of a list of named lists/classes fulfill a certain condition. If not, an error is issued and all elements of the list not fulfilling the condition are listed.

Usage

```
assert_list_element(list, element, condition, message_text, ...)
```

Arguments

`list`	A list to be checked A list of named lists or classes is expected.
`element`	The name of an element of the lists/classes A character scalar is expected.
`condition`	Condition to be fulfilled The condition is evaluated for each element of the list. The element of the lists/classes can be referred to by its name, e.g., `censor == 0` to check the `censor` field of a class.
`message_text`	Text to be displayed in the message The text should describe the condition to be fulfilled, e.g., "For events the censor values must be zero.".
`...`	Objects required to evaluate the condition If the condition contains objects apart from the element, they have to be passed to the function. See the second example below.

Value

An error if the condition is not meet. The input otherwise.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

assert_list_of — *Is an Argument a List of Objects of a Specific S3 Class or Type?*

Description

Checks if an argument is a `list` of objects inheriting from the S3 class or type specified.

Usage

```
assert_list_of(arg, class, named = FALSE, optional = TRUE)
```

Arguments

`arg`	A function argument to be checked
`class`	The S3 class or type to check for
`named`	If set to `TRUE`, an error is issued if not all elements of the list are named.
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a list or if `arg` is a list but its elements are not objects inheriting from `class` or of type `class`. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(list) {
  assert_list_of(list, "data.frame")
}

example_fun(list(mtcars, iris))

try(example_fun(list(letters, 1:10)))

try(example_fun(c(TRUE, FALSE)))

example_fun2 <- function(list) {
  assert_list_of(list, "numeric", named = TRUE)
}
try(example_fun2(list(1, 2, 3, d = 4)))
```

`assert_logical_scalar` *Is an Argument a Logical Scalar (Boolean)?*

Description

Checks if an argument is a logical scalar

Usage

```
assert_logical_scalar(arg, optional = FALSE)
```

Arguments

`arg`	A function argument to be checked
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown. Otherwise, `NULL` is considered as valid value.

Value

The function throws an error if `arg` is neither `TRUE` or `FALSE`. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_ex` `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(flag) {
  assert_logical_scalar(flag)
}

example_fun(FALSE)

try(example_fun(NA))

try(example_fun(c(TRUE, FALSE, FALSE)))

try(example_fun(1:10))
```

`assert_named` *Assert Argument is a Named List or Vector*

Description

Assert that all elements of the argument are named.

Usage

```
assert_named(arg, optional = FALSE)
```

Arguments

`arg`	A function argument to be checked
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a named list or vector or returns the input invisibly otherwise

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(varval_list) {
  assert_named(varval_list)
}

example_fun(list(var1 = 1, var2 = "x"))

try(example_fun(list(1, "x")))

try(example_fun(list(var = 1, "x")))
```

`assert_named_exprs` *Assert Argument is a Named List of Expressions*

Description

[Deprecated]

This function is *deprecated*, please use `assert_expr_list()` instead.

Usage

```
assert_named_exprs(arg, optional = FALSE)
```

Arguments

`arg`	A function argument to be checked
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a named `list` of expression or returns the input invisibly otherwise

See Also

Other deprecated: `assert_function_param()`, `assert_has_variables()`

`assert_numeric_vector` *Is an Argument a Numeric Vector?*

Description

Checks if an argument is a numeric vector

Usage

```
assert_numeric_vector(arg, optional = FALSE)
```

Arguments

`arg`	A function argument to be checked
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a numeric vector. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_one_to_one()`, `assert_param_does_not_ex`
`assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(num) {
  assert_numeric_vector(num)
}

example_fun(1:10)

try(example_fun(letters))
```

`assert_one_to_one` *Is There a One to One Mapping between Variables?*

Description

Checks if there is a one to one mapping between two lists of variables.

Usage

```
assert_one_to_one(dataset, vars1, vars2)
```

Arguments

`dataset`	Dataset to be checked The variables specified for `vars1` and `vars2` are expected.
`vars1`	First list of variables
`vars2`	Second list of variables

Value

An error if the condition is not meet. The input otherwise.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

`assert_param_does_not_exist` *Asserts That a Parameter Does Not Exist in the Dataset*

Description

Checks if a parameter (`PARAMCD`) does not exist in a dataset.

Usage

```
assert_param_does_not_exist(dataset, param)
```

Arguments

`dataset`	A `data.frame`
`param`	Parameter code to check

Value

The function throws an error if the parameter exists in the input dataset. Otherwise, the dataset is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
library(tibble)
advs <- tribble(
  ~USUBJID, ~VSTESTCD, ~VSSTRESN, ~VSSTRESU, ~PARAMCD, ~AVAL,
  "P01",    "WEIGHT",      80.1, "kg",      "WEIGHT",  80.1,
  "P02",    "WEIGHT",      85.7, "kg",      "WEIGHT",  85.7
)
assert_param_does_not_exist(advs, param = "HR")
try(assert_param_does_not_exist(advs, param = "WEIGHT"))
```

assert_s3_class *Is an Argument an Object of a Specific S3 Class?*

Description

Checks if an argument is an object inheriting from the S3 class specified.

Usage

```
assert_s3_class(arg, class, optional = FALSE)
```

Arguments

arg	A function argument to be checked
class	The S3 class to check for
optional	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is an object which does *not* inherit from `class`. Otherwise, the input is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(obj) {
  assert_s3_class(obj, "factor")
}

example_fun(as.factor(letters))

try(example_fun(letters))

try(example_fun(1:10))
```

`assert_same_type` *Are All Argument of the Same Type?*

Description

Checks if all arguments are of the same type.

Usage

```
assert_same_type(...)
```

Arguments

`...`	Arguments to be checked

Value

The function throws an error if not all arguments are of the same type.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
example_fun <- function(true_value, false_value, missing_value) {
  assert_same_type(true_value, false_value, missing_value)
}

example_fun(
  true_value = "Y",
  false_value = "N",
  missing_value = NA_character_
)

try(example_fun(
  true_value = 1,
  false_value = 0,
  missing_value = "missing"
))
```

assert_symbol *Is an Argument a Symbol?*

Description

Checks if an argument is a symbol

Usage

```
assert_symbol(arg, optional = FALSE)
```

Arguments

`arg`	A function argument to be checked. Must be a `symbol`. See examples.
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a symbol and returns the input invisibly otherwise.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
library(pharmaversesdtm)
library(dplyr, warn.conflicts = FALSE)
library(rlang)
data(dm)

example_fun <- function(dat, var) {
  var <- assert_symbol(enexpr(var))
  select(dat, !!var)
}

example_fun(dm, USUBJID)

try(example_fun(dm))

try(example_fun(dm, "USUBJID"))

try(example_fun(dm, toupper(PARAMCD)))
```

assert_unit *Asserts That a Parameter is Provided in the Expected Unit*

Description

Checks if a parameter (`PARAMCD`) in a dataset is provided in the expected unit.

Usage

```
assert_unit(dataset, param, required_unit, get_unit_expr)
```

Arguments

`dataset`	A `data.frame`
`param`	Parameter code of the parameter to check
`required_unit`	Expected unit
`get_unit_expr`	Expression used to provide the unit of `param`

Value

The function throws an error if the unit variable differs from the unit for any observation of the parameter in the input dataset. Otherwise, the dataset is returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_vars()`, `assert_varval_list()`

Examples

```
library(tibble)
advs <- tribble(
  ~USUBJID, ~VSTESTCD, ~VSTRESN, ~VSSTRESU, ~PARAMCD, ~AVAL,
  "P01",    "WEIGHT",      80.1, "kg",      "WEIGHT",  80.1,
  "P02",    "WEIGHT",      85.7, "kg",      "WEIGHT",  85.7
)

assert_unit(advs, param = "WEIGHT", required_unit = "kg", get_unit_expr = VSSTRESU)
```

assert_vars *Is an Argument a List of Variables?*

Description

Checks if an argument is a valid list of symbols (e.g., created by `exprs()`)

Usage

```
assert_vars(arg, expect_names = FALSE, optional = FALSE)
```

Arguments

`arg`	A function argument to be checked
`expect_names`	If the argument is set to `TRUE`, it is checked if all variables are named, e.g., `exprs(APERSDT = APxxSDT, APEREDT = APxxEDT)`.
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown

Value

The function throws an error if `arg` is not a list of symbols (e.g., created by `exprs()` and returns the input invisibly otherwise.

See Also

Checks for valid input and returns warning or errors messages: `assert_atomic_vector()`, `assert_character_scalar()`, `assert_character_vector()`, `assert_data_frame()`, `assert_date_vector()`, `assert_expr_list()`, `assert_expr()`, `assert_filter_cond()`, `assert_function()`, `assert_integer_scalar()`, `assert_list_element()`, `assert_list_of()`, `assert_logical_scalar()`, `assert_named()`, `assert_numeric_vector()`, `assert_one_to_one()`, `assert_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_varval_list()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(rlang)

example_fun <- function(by_vars) {
  assert_vars(by_vars)
}

example_fun(exprs(USUBJID, PARAMCD))

try(example_fun(quos(USUBJID, PARAMCD)))

try(example_fun(c("USUBJID", "PARAMCD", "VISIT")))
try(example_fun(exprs(USUBJID, toupper(PARAMCD), desc(AVAL))))

example_fun_name <- function(by_vars) {
  assert_vars(by_vars, expect_names = TRUE)
}

example_fun_name(exprs(APERSDT = APxxSDT, APEREDT = APxxEDT))

try(example_fun_name(exprs(APERSDT = APxxSDT, APxxEDT)))
```

`assert_varval_list` *Is an Argument a Variable-Value List?*

Description

Checks if the argument is a list of expressions where the expressions are variable-value pairs. The value can be a symbol, a string, a numeric, an expression, or `NA`.

Usage

```
assert_varval_list(
  arg,
  required_elements = NULL,
  accept_expr = TRUE,
  accept_var = FALSE,
  optional = FALSE
)
```

Arguments

`arg`	A function argument to be checked
`required_elements`	A `character` vector of names that must be present in `arg`
`accept_expr`	Should expressions on the right hand side be accepted?
`accept_var`	Should unnamed variable names (e.g. `exprs(USUBJID)`) on the right hand side be accepted?
`optional`	Is the checked argument optional? If set to `FALSE` and `arg` is `NULL` then an error is thrown.

Value

The function throws an error if `arg` is not a list of variable-value expressions. Otherwise, the input it returned invisibly.

See Also

Checks for valid input and returns warning or errors messages: assert_atomic_vector(), assert_character_scalar(), assert_character_vector(), assert_data_frame(), assert_date_vector(), assert_expr_list(), assert_expr(), assert_filter_cond(), assert_function(), assert_integer_scalar(), assert_list_element(), assert_list_of(), assert_logical_scalar(), assert_named(), assert_numeric_vector(), assert_one_to_one(), assert_param_does_not_exist(), assert_s3_class(), assert_same_type(), assert_symbol(), assert_unit(), assert_vars()

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(rlang)

example_fun <- function(vars) {
  assert_varval_list(vars)
}
example_fun(exprs(DTHDOM = "AE", DTHSEQ = AESEQ))

try(example_fun(exprs("AE", DTSEQ = AESEQ)))
```

---

backquote *Wrap a String in Backquotes*

Description

Wrap a String in Backquotes

Usage

```
backquote(x)
```

Arguments

x	A character vector

Value

A character vector

See Also

Helpers for working with Quotes and Quoting: dquote(), enumerate(), squote()

`contains_vars` *check that argument contains valid variable(s) created with* `exprs()` *or Source Variables from a List of Expressions*

Description

check that argument contains valid variable(s) created with `exprs()` or Source Variables from a List of Expressions

Usage

```
contains_vars(arg)
```

Arguments

`arg`	A function argument to be checked

Value

A `TRUE` if variables were valid variable

See Also

Developer Utility Functions: `%notin%()`, `%or%()`, `arg_name()`, `convert_dtm_to_dtc()`, `extract_vars()`, `filter_if()`, `friendly_type_of()`, `valid_time_units()`, `vars2chr()`

`convert_dtm_to_dtc` *Helper Function to Convert Date (or Date-time) Objects to Characters of dtc Format (-DTC type of variable)*

Description

Helper Function to Convert Date (or Date-time) Objects to Characters of dtc Format (-DTC type of variable)

Usage

```
convert_dtm_to_dtc(dtm)
```

Arguments

`dtm`	date or date-time

Value

`character` vector

See Also

Developer Utility Functions: `%notin%()`, `%or%()`, `arg_name()`, `contains_vars()`, `extract_vars()`, `filter_if()`, `friendly_type_of()`, `valid_time_units()`, `vars2chr()`

`dataset_vignette` *Output a Dataset in a Vignette in the admiral Format*

Description

Output a dataset in a vignette with the pre-specified admiral format.

Usage

```
dataset_vignette(dataset, display_vars = NULL, filter = NULL)
```

Arguments

`dataset`	Dataset to output in the vignette
`display_vars`	Variables selected to demonstrate the outcome of the derivation Permitted Values: list of variables Default is NULL If `display_vars` is not NULL, only the selected variables are visible in the vignette while the other variables are hidden. They can be made visible by clicking the`Choose the columns to display` button.
`filter`	Filter condition The specified condition is applied to the dataset before it is displayed. Permitted Values: a condition

Value

A HTML table

`dquote` *Wrap a String in Double Quotes*

Description

Wrap a string in double quotes, e.g., for displaying character values in messages.

Usage

```
dquote(x)
```

Arguments

`x`	A character vector

Value

If the input is `NULL`, the text `"NULL"` is returned. Otherwise, the input in double quotes is returned.

See Also

Helpers for working with Quotes and Quoting: `backquote()`, `enumerate()`, `squote()`

`enumerate` *Enumerate Multiple Elements*

Description

Enumerate multiple elements of a vector or list.

Usage

```
enumerate(x, quote_fun = backquote, conjunction = "and")
```

Arguments

`x`	A vector or list
`quote_fun`	Quoting function, defaults to `backquote`. If set to `NULL`, the elements are not quoted.
`conjunction`	Character to be used in the message, defaults to `"and"`.

Value

A `character` vector

See Also

Helpers for working with Quotes and Quoting: `backquote()`, `dquote()`, `squote()`

Examples

```
enumerate(c("one", "two", "three"))

enumerate(c(1, 2, 3), quote_fun = NULL)
```

expect_dfs_equal *Expectation: Are Two Datasets Equal?*

Description

Uses `diffdf::diffdf()` to compares 2 datasets for any differences. This function can be thought of as an R-equivalent of SAS proc compare and a useful tool for unit testing as well.

Usage

```
expect_dfs_equal(base, compare, keys, ...)
```

Arguments

`base`	Input dataset
`compare`	Comparison dataset
`keys`	`character` vector of variables that define a unique row in the `base` and `compare` datasets
`...`	Additional arguments passed onto `diffdf::diffdf()`

Value

An error if `base` and `compare` do not match or `NULL` invisibly if they do

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(tibble)

tbl1 <- tribble(
  ~USUBJID, ~AGE, ~SEX,
  "1001", 18, "M",
  "1002", 19, "F",
  "1003", 20, "M",
  "1004", 18, "F"
)

tbl2 <- tribble(
  ~USUBJID, ~AGE, ~SEX,
  "1001", 18, "M",
  "1002", 18.9, "F",
  "1003", 20, NA
)

try(expect_dfs_equal(tbl1, tbl2, keys = "USUBJID"))

tlb3 <- tribble(
  ~USUBJID, ~AGE, ~SEX,
  "1004", 18, "F",
  "1003", 20, "M",
  "1002", 19, "F",
  "1001", 18, "M",
)

# Note the sorting order of the keys is not required
expect_dfs_equal(tbl1, tlb3, keys = "USUBJID")
```

---

expr_c — *Concatenate One or More Expressions*

Description

Concatenate One or More Expressions

Usage

```
expr_c(...)
```

Arguments

`...`	One or more expressions or list of expressions

Value

A list of expressions

See Also

Helpers for working with Quosures: `add_suffix_to_vars()`, `replace_symbol_in_expr()`, `replace_values_by_names(`

---

extract_vars — *Extract All Symbols from a List of Expressions*

Description

Extract All Symbols from a List of Expressions

Usage

```
extract_vars(x, side = "lhs")
```

Arguments

`x`	An R object
`side`	One of `"lhs"` (the default) or `"rhs"` for formulas

Value

A list of expressions

See Also

Developer Utility Functions: %notin%(), %or%(), arg_name(), contains_vars(), convert_dtm_to_dtc(), filter_if(), friendly_type_of(), valid_time_units(), vars2chr()

Examples

```
library(rlang)
extract_vars(exprs(PARAMCD, (BASE - AVAL) / BASE + 100))
extract_vars(AVAL ~ ARMCD + AGEGR1)
extract_vars(AVAL ~ ARMCD + AGEGR1, side = "rhs")
```

---

filter_if *Optional Filter*

Description

Filters the input dataset if the provided expression is not NULL

Usage

```
filter_if(dataset, filter)
```

Arguments

dataset	Input dataset
filter	A filter condition. Must be an expression.

Value

A data.frame containing all rows in dataset matching filter or just dataset if filter is NULL

See Also

Developer Utility Functions: %notin%(), %or%(), arg_name(), contains_vars(), convert_dtm_to_dtc(), extract_vars(), friendly_type_of(), valid_time_units(), vars2chr()

`friendly_type_of` *Return English-friendly messaging for object-types*

Description

Return English-friendly messaging for object-types

Usage

```
friendly_type_of(x, value = TRUE, length = FALSE)
```

Arguments

`x`	Any R object.
`value`	Whether to describe the value of `x`.
`length`	Whether to mention the length of vectors and lists.

Details

This helper function aids us in forming user-friendly messages that gets called through `what_is_it()`, which is often used in the assertion functions to identify what object-type the user passed through an argument instead of an expected-type.

Value

A string describing the type. Starts with an indefinite article, e.g. "an integer vector".

See Also

Developer Utility Functions: `%notin%()`, `%or%()`, `arg_name()`, `contains_vars()`, `convert_dtm_to_dtc()`, `extract_vars()`, `filter_if()`, `valid_time_units()`, `vars2chr()`

`get_constant_vars` *Get Constant Variables*

Description

Get Constant Variables

Usage

```
get_constant_vars(dataset, by_vars, ignore_vars = NULL)
```

Arguments

`dataset`	A data frame.
`by_vars`	By variables The groups defined by the by variables are considered separately. I.e., if a variable is constant within each by group, it is returned.
`ignore_vars`	Variables to ignore The specified variables are not considered, i.e., they are not returned even if they are constant (unless they are included in the by variables). *Permitted Values:* A list of variable names or selector function calls like `starts_with("EX")`

Value

Variable vector.

See Also

Brings something to you!?!: `get_dataset()`, `get_duplicates()`, `get_source_vars()`

`get_dataset` *Retrieve a Dataset from the* `admiraldev_environment` *environment*

Description

Retrieve a Dataset from the `admiraldev_environment` environment

Usage

```
get_dataset(name)
```

Arguments

`name`	The name of the dataset to retrieve

Details

Sometimes, developers may want to provide information to users which does not fit into a warning or error message. For example, if the input dataset of a function contains unexpected records, these can be stored in a separate dataset, which users can access to investigate the issue.

To achieve this, R has a data structure known as an 'environment'. These environment objects are created at build time, but can be populated with values after the package has been loaded and update those values over the course of an R session.

As so, the establishment of `admiraldev_environment` allows us to create dynamic data/objects based on user-inputs that need modification. The purpose of `get_dataset` is to retrieve the datasets contained inside `admiraldev_environment`.

Currently we only support two datasets inside our `admiraldev_environment` object:

- `one_to_many`
- `many_to_one`

Value

A `data.frame`

See Also

Brings something to you!?!: `get_constant_vars()`, `get_duplicates()`, `get_source_vars()`

`get_duplicates` *Get Duplicates From a Vector*

Description

Get Duplicates From a Vector

Usage

```
get_duplicates(x)
```

Arguments

`x`	An atomic vector

Value

A vector of the same type as `x` contain duplicate values

See Also

Brings something to you!?!: `get_constant_vars()`, `get_dataset()`, `get_source_vars()`

Examples

```
get_duplicates(1:10)

get_duplicates(c("a", "a", "b", "c", "d", "d"))
```

get_new_tmp_var — *Get a New Temporary Variable Name for a Dataset*

Description

Get a New Temporary Variable Name for a Dataset

Usage

```
get_new_tmp_var(dataset, prefix = "tmp_var")
```

Arguments

`dataset`	The input dataset
`prefix`	The prefix of the new temporary variable name to create

Details

The function returns a new unique temporary variable name to be used inside `dataset`. The temporary variable names have the structure `prefix_n` where `n` is an integer, e.g. `tmp_var_1`. If there is already a variable inside `datset` with a given `prefix` then the suffix is increased by 1, e.g. if `tmp_var_1` already exists then `get_new_tmp_var()` will return `tmp_var_2`.

Value

The name of a new temporary variable as a symbol

See Also

`remove_tmp_vars()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(pharmaversesdtm)
data(dm)

tmp_var <- get_new_tmp_var(dm)
mutate(dm, !!tmp_var := NA)
```

get_source_vars *Get Source Variables from a List of Expressions*

Description

Get Source Variables from a List of Expressions

Usage

```
get_source_vars(expressions, quosures)
```

Arguments

expressions	A list of expressions
quosures	*Deprecated*, please use expressions instead.

Value

A list of expressions

See Also

Brings something to you!?!: get_constant_vars(), get_dataset(), get_duplicates()

is_auto *Checks if the argument equals the auto keyword*

Description

Checks if the argument equals the auto keyword

Usage

```
is_auto(arg)
```

Arguments

arg	argument to check

Value

TRUE if the argument equals the auto keyword, i.e., it is an expression of a symbol named auto.

See Also

Identifies type of Object with return of TRUE/FALSE: is_named(), is_order_vars(), is_valid_dtc()

`is_named` *Is a named argument*

Description

Is a named argument

Usage

```
is_named(x)
```

Arguments

`x`	Any R object

Value

`TRUE` if the argument is named, `FALSE` otherwise

See Also

Identifies type of Object with return of TRUE/FALSE: `is_auto()`, `is_order_vars()`, `is_valid_dtc()`

`is_order_vars` *Is order vars?*

Description

Check if inputs are created using `exprs()` or calls involving `desc()`

Usage

```
is_order_vars(arg)
```

Arguments

`arg`	An R object

Value

`FALSE` if the argument is not a list of order vars

See Also

Identifies type of Object with return of TRUE/FALSE: `is_auto()`, `is_named()`, `is_valid_dtc()`

`is_valid_dtc` *Is this string a valid DTC*

Description

Is this string a valid DTC

Usage

```
is_valid_dtc(arg)
```

Arguments

`arg`	A `character` vector

Value

`TRUE` if the argument is a valid `--DTC` string, `FALSE` otherwise

See Also

Identifies type of Object with return of TRUE/FALSE: `is_auto()`, `is_named()`, `is_order_vars()`

`process_set_values_to` *Process* `set_values_to` *Argument*

Description

The function creates the variables specified by the `set_values_to` argument, catches errors, provides user friendly error messages, and optionally checks the type of the created variables.

Usage

```
process_set_values_to(dataset, set_values_to = NULL, expected_types = NULL)
```

Arguments

`dataset`	Input dataset
`set_values_to`	Variables to set A named list returned by `exprs()` defining the variables to be set, e.g. `exprs(PARAMCD = "OS", PARAM = "Overall Survival")` is expected. The values must be symbols, character strings, numeric values, expressions, or `NA`.
`expected_types`	If the argument is specified, the specified variables are checked whether the specified type matches the type of the variables created by `set_values_to`. *Permitted Values*: A character vector with values `"numeric"` or `"character"`

Value

The input dataset with the variables specified by `set_values_to` added/updated

Examples

```
library(tibble)
data <- tribble(
  ~AVAL,
  20
)

try(
  process_set_values_to(
    data,
    set_values_to = exprs(
      PARAMCD = BMI
    )
  )
)

try(
  process_set_values_to(
    data,
    set_values_to = exprs(
      PARAMCD = 42
    ),
    expected_types = c(PARAMCD = "character")
  )
)
```

`remove_tmp_vars` *Remove All Temporary Variables Created Within the Current Function Environment*

Description

Remove All Temporary Variables Created Within the Current Function Environment

Usage

```
remove_tmp_vars(dataset)
```

Arguments

`dataset`	The input dataset

Value

The input dataset with temporary variables removed

See Also

get_new_tmp_var()

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(pharmaversesdtm)
data(dm)
dm <- select(dm, USUBJID)
tmp_var <- get_new_tmp_var(dm)
dm <- mutate(dm, !!tmp_var := NA)

## This function creates two new temporary variables which are removed when calling
## `remove_tmp_vars()`. Note that any temporary variable created outside this
## function is **not** removed
do_something <- function(dataset) {
  tmp_var_1 <- get_new_tmp_var(dm)
  tmp_var_2 <- get_new_tmp_var(dm)
  dm %>%
    mutate(!!tmp_var_1 := NA, !!tmp_var_2 := NA) %>%
    print() %>%
    remove_tmp_vars()
}

do_something(dm)
```

---

replace_symbol_in_expr
Replace Symbols in an Expression

Description

Replace symbols in an expression

Usage

```
replace_symbol_in_expr(expression, target, replace)
```

Arguments

expression	Expression
target	Target symbol
replace	Replacing symbol

Value

The expression where every occurrence of the symbol `target` is replaced by `replace`

Author(s)

Stefan Bundfuss

See Also

Helpers for working with Quosures: `add_suffix_to_vars()`, `expr_c()`, `replace_values_by_names()`

Examples

```
library(rlang)

replace_symbol_in_expr(expr(AVAL), target = AVAL, replace = AVAL.join)
replace_symbol_in_expr(expr(AVALC), target = AVAL, replace = AVAL.join)
replace_symbol_in_expr(expr(desc(AVAL)), target = AVAL, replace = AVAL.join)
```

---

replace_values_by_names

Replace Expression Value with Name

Description

Replace Expression Value with Name

Usage

```
replace_values_by_names(expressions, quosures)
```

Arguments

`expressions`	A list of expressions
`quosures`	*Deprecated*, please use `expressions` instead.

Value

A list of expressions

See Also

Helpers for working with Quosures: `add_suffix_to_vars()`, `expr_c()`, `replace_symbol_in_expr()`

Examples

```
library(rlang)
replace_values_by_names(exprs(AVAL, ADT = convert_dtc_to_dt(EXSTDTC)))
```

`squote` *Wrap a String in Single Quotes*

Description

Wrap a String in Single Quotes

Usage

```
squote(x)
```

Arguments

`x`	A `character` vector

Value

A `character` vector

See Also

Helpers for working with Quotes and Quoting: `backquote()`, `dquote()`, `enumerate()`

`suppress_warning` *Suppress Specific Warnings*

Description

Suppress certain warnings issued by an expression.

Usage

```
suppress_warning(expr, regexpr)
```

Arguments

`expr`	Expression to be executed
`regexpr`	Regular expression matching warnings to suppress

Details

All warnings which are issued by the expression and match the regular expression are suppressed.

Value

Return value of the expression

See Also

Function that provide users with custom warnings `warn_if_incomplete_dtc()`, `warn_if_inconsistent_list()`, `warn_if_invalid_dtc()`, `warn_if_vars_exist()`

`valid_time_units` *Valid Time Units*

Description

Contains the acceptable character vector of valid time units

Usage

```
valid_time_units()
```

Value

A `character` vector of valid time units

See Also

Developer Utility Functions: `%notin%()`, `%or%()`, `arg_name()`, `contains_vars()`, `convert_dtm_to_dtc()`, `extract_vars()`, `filter_if()`, `friendly_type_of()`, `vars2chr()`

`vars2chr` *Turn a List of Expressions into a Character Vector*

Description

Turn a List of Expressions into a Character Vector

Usage

```
vars2chr(expressions, quosures)
```

Arguments

`expressions`	A `list` of expressions created using `exprs()`
`quosures`	*Deprecated*, please use `expressions` instead.

Value

A character vector

See Also

Developer Utility Functions: `%notin%()`, `%or%()`, `arg_name()`, `contains_vars()`, `convert_dtm_to_dtc()`, `extract_vars()`, `filter_if()`, `friendly_type_of()`, `valid_time_units()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(rlang)

vars2chr(exprs(USUBJID, AVAL))
```

---

warn_if_incomplete_dtc   *Warn if incomplete dtc*

Description

Warn if incomplete dtc

Usage

```
warn_if_incomplete_dtc(dtc, n)
```

Arguments

`dtc`	A `character` vector of date-times in ISO 8601 format
`n`	A non-negative integer

Value

A warning if `dtc` contains any partial dates

See Also

Function that provide users with custom warnings `suppress_warning()`, `warn_if_inconsistent_list()`, `warn_if_invalid_dtc()`, `warn_if_vars_exist()`

`warn_if_inconsistent_list` *Warn If Two Lists are Inconsistent*

Description

Checks if two list inputs have the same names and same number of elements and issues a warning otherwise.

Usage

```
warn_if_inconsistent_list(base, compare, list_name, i = 2)
```

Arguments

`base`	A named list
`compare`	A named list
`list_name`	A string the name of the list
`i`	the index id to compare the 2 lists

Value

a `warning` if the 2 lists have different names or length

See Also

Function that provide users with custom warnings `suppress_warning()`, `warn_if_incomplete_dtc()`, `warn_if_invalid_dtc()`, `warn_if_vars_exist()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(rlang)

# no warning
warn_if_inconsistent_list(
  base = exprs(DTHDOM = "DM", DTHSEQ = DMSEQ),
  compare = exprs(DTHDOM = "DM", DTHSEQ = DMSEQ),
  list_name = "Test"
)
# warning
warn_if_inconsistent_list(
  base = exprs(DTHDOM = "DM", DTHSEQ = DMSEQ, DTHVAR = "text"),
  compare = exprs(DTHDOM = "DM", DTHSEQ = DMSEQ),
  list_name = "Test"
)
```

`warn_if_invalid_dtc` *Warn If a Vector Contains Unknown Datetime Format*

Description

Warn if the vector contains unknown datetime format such as "2003-12-15T-:15:18", "2003-12-15T13:-:19","–12-15","——T07:15"

Usage

```
warn_if_invalid_dtc(dtc, is_valid = is_valid_dtc(dtc))
```

Arguments

`dtc`	a character vector containing the dates
`is_valid`	a logical vector indicating whether elements in `dtc` are valid

Value

No return value, called for side effects

See Also

Function that provide users with custom warnings `suppress_warning()`, `warn_if_incomplete_dtc()`, `warn_if_inconsistent_list()`, `warn_if_vars_exist()`

Examples

```
## No warning as `dtc` is a valid date format
warn_if_invalid_dtc(dtc = "2021-04-06")

## Issues a warning
warn_if_invalid_dtc(dtc = "2021-04-06T-:30:30")
```

`warn_if_vars_exist` *Warn If a Variable Already Exists*

Description

Warn if a variable already exists inside a dataset

Usage

```
warn_if_vars_exist(dataset, vars)
```

Arguments

`dataset`	A `data.frame`
`vars`	`character` vector of columns to check for in `dataset`

Value

No return value, called for side effects

See Also

Function that provide users with custom warnings `suppress_warning()`, `warn_if_incomplete_dtc()`, `warn_if_inconsistent_list()`, `warn_if_invalid_dtc()`

Examples

```
library(pharmaversesdtm)
data(dm)

## No warning as `AAGE` doesn't exist in `dm`
warn_if_vars_exist(dm, "AAGE")

## Issues a warning
warn_if_vars_exist(dm, "ARM")
```

`what_is_it` *What Kind of Object is This?*

Description

Returns a string describing what kind of object the input is.

Usage

```
what_is_it(x)
```

Arguments

`x`	Any R object

Value

A `character` description of the type of `x`

Examples

```
what_is_it("abc")
what_is_it(1L)
what_is_it(1:10)
what_is_it(mtcars)
```

`%notin%` *Negated Value Matching*

Description

Returns a `logical` vector indicating if there is *no* match of the left operand in the right operand.

Usage

```
x %notin% table
```

Arguments

`x`	The values to be matched
`table`	The values to be matched against

Value

A `logical` vector

See Also

Developer Utility Functions: `%or%()`, `arg_name()`, `contains_vars()`, `convert_dtm_to_dtc()`, `extract_vars()`, `filter_if()`, `friendly_type_of()`, `valid_time_units()`, `vars2chr()`

`%or%` *Or*

Description

Or

Usage

```
lhs %or% rhs
```

Arguments

`lhs`	Any valid R expression
`rhs`	Any valid R expression

Details

The function evaluates the expression `lhs` and if this expression results in an error, it catches that error and proceeds with evaluating the expression `rhs` and returns that result.

Value

Either the result of evaluating `lhs`, `rhs` or an error

See Also

Developer Utility Functions: `%notin%()`, `arg_name()`, `contains_vars()`, `convert_dtm_to_dtc()`, `extract_vars()`, `filter_if()`, `friendly_type_of()`, `valid_time_units()`, `vars2chr()`

Index