

Package ‘admiraldev’

June 10, 2026

Type Package

Title Utility Functions and Development Tools for the Admiral Package Family

Version 1.5.0

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)

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

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

Depends R (≥ 4.1)

Imports cli ($\geq 3.6.2$), dplyr ($\geq 1.1.1$), glue ($\geq 1.6.0$), lifecycle ($\geq 0.1.0$), lubridate ($\geq 1.7.4$), purrr ($\geq 0.3.3$), rlang ($\geq 0.4.4$), roxygen2 ($\geq 8.0.0$), stringr ($\geq 1.4.0$), tidyr ($\geq 1.0.2$), tidyselect ($\geq 1.1.0$), withr

Suggests diffdf, DT, htmltools, knitr, methods, pkgdown, rmarkdown, spelling, testthat ($\geq 3.2.0$)

VignetteBuilder knitr

Config/roxygen2/markdown TRUE

Config/roxygen2/roclats collate, namespace, admiraldev::rdx_roclet

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

NeedsCompilation no

Author Edoardo Mancini [aut, cre] (ORCID: <https://orcid.org/0009-0006-4899-8641>),
 Stefan Bundfuss [aut] (ORCID: <https://orcid.org/0009-0005-0027-1198>),
 Arianna Cascone [aut] (ORCID: <https://orcid.org/0000-0001-5948-2831>),
 Kristin Dahnert [aut],
 Jeffrey Dickinson [aut],
 Ross Farrugia [aut],
 Fanny Gautier [aut] (ORCID: <https://orcid.org/0009-0004-3581-0131>),
 Liam Hobby [aut],
 Gordon Miller [aut],
 Lina Patil [aut],
 Ben Straub [aut],
 F. Hoffmann-La Roche AG [cph, fnd],
 GlaxoSmithKline LLC [cph, fnd]

Maintainer Edoardo Mancini <edoardo.mancini@roche.com>

Repository CRAN

Date/Publication 2026-06-10 09:40:08 UTC

Contents

add_suffix_to_vars	4
arg_name	5
assert_atomic_vector	5
assert_character_scalar	7
assert_character_vector	9
assert_data_frame	10
assert_date_var	12
assert_date_vector	14
assert_expr	16
assert_expr_list	17
assert_filter_cond	19
assert_function	21
assert_integer_scalar	22
assert_list_element	24
assert_list_of	26
assert_logical_scalar	28
assert_named	30
assert_numeric_vector	31
assert_one_to_one	33
assert_param_does_not_exist	35
assert_s3_class	36
assert_same_type	38
assert_symbol	39
assert_unit	41
assert_vars	43
assert_varval_list	45
backquote	47

capture_output	47
contains_vars	48
convert_dtm_to_dtc	49
dataset_vignette	50
deprecate_inform	50
dquote	52
enumerate	53
expect_dfs_equal	53
expr_c	55
extract_vars	55
filter_if	56
friendly_type_of	57
get_constant_vars	58
get_dataset	58
get_duplicates	59
get_new_tmp_var	60
get_source_vars	61
is_auto	61
is_order_vars	62
is_valid_dtc	63
parse_code	64
process_set_values_to	65
rdx_roclet	66
remove_tmp_vars	68
replace_symbol_in_expr	70
replace_values_by_names	71
roxygen_order_na_handling	71
roxygen_param_by_vars	72
roxygen_param_dataset	73
roxygen_save_memory	74
squote	74
suppress_warning	75
valid_time_units	76
vars2chr	76
warn_if_incomplete_dtc	77
warn_if_inconsistent_list	78
warn_if_invalid_dtc	79
warn_if_vars_exist	80
what_is_it	81
%notin%	81
%or%	82

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)) Default value none
vars	Variables to change Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT) Default value none
suffix	Suffix Permitted values a character scalar, i.e., a character vector of length one Default value none

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")
```

arg_name	<i>Extract Argument Name from an Expression</i>
----------	---

Description**[Deprecated]**

This function is *deprecated*, please use `rlang::caller_arg()` instead.

Usage

```
arg_name(expr)
```

Arguments

expr An expression created inside a function using `substitute()`
Default value none

Value

character vector

See Also

Other deprecated: [%or%](#), [enumerate\(\)](#), [friendly_type_of\(\)](#), [valid_time_units\(\)](#), [what_is_it\(\)](#)

assert_atomic_vector	<i>Is an Argument an Atomic Vector?</i>
----------------------	---

Description

Checks if an argument is an atomic vector

Usage

```
assert_atomic_vector(  
  arg,  
  optional = FALSE,  
  arg_name = rlang::caller_arg(arg),  
  message = NULL,  
  class = "assert_atomic_vector",  
  call = parent.frame()  
)
```

Arguments

arg	A function argument to be checked Default value none
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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\(\)](#), [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\(\)](#)

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,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_character_scalar",
  call = parent.frame()
)
```

Arguments

arg	A function argument to be checked Default value none
values	A character vector of valid values for arg. Values is converted to a lower case vector if case_sensitive = FALSE is used. Default value NULL
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. Default value TRUE
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL

class	Subclass of the condition.
call	<p>The execution environment of a currently running function, e.g. <code>call = caller_env()</code>. The corresponding function call is retrieved and mentioned in error messages as the source of the error.</p> <p>You only need to supply <code>call</code> when throwing a condition from a helper function which wouldn't be relevant to mention in the message.</p> <p>Can also be <code>NULL</code> or a defused function call to respectively not display any call or hard-code a code to display.</p> <p>For more information about error calls, see Including function calls in error messages.</p>

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()`, `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()`

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,  
  arg_name = rlang::caller_arg(arg),  
  message = NULL,  
  class = "assert_character_vector",  
  call = parent.frame()  
)
```

Arguments

<code>arg</code>	A function argument to be checked Default value none
<code>values</code>	A character vector of valid values for <code>arg</code> Default value NULL
<code>named</code>	If set to TRUE, an error is issued if not all elements of the vector are named. Default value FALSE
<code>optional</code>	Is the checked argument optional? If set to FALSE and <code>arg</code> is NULL then an error is thrown Default value FALSE
<code>arg_name</code>	string indicating the label/symbol of the object being checked. Default value <code>rlang::caller_arg(arg)</code>
<code>message</code>	string passed to <code>cli::cli_abort(message)</code> . When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
<code>class</code>	Subclass of the condition.

`call` The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error.

You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message.

Can also be `NULL` or a [defused function call](#) to respectively not display any call or hard-code a code to display.

For more information about error calls, see [Including function calls in error messages](#).

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\(\)](#), [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\(\)](#)

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 it contains a set of required variables

Usage

```

assert_data_frame(
  arg,
  required_vars = NULL,
  check_is_grouped = TRUE,
  check_is_rowwise = TRUE,
  optional = FALSE,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_data_frame",
  call = parent.frame()
)

```

Arguments

arg	A function argument to be checked Default value none
required_vars	A list of variables created using <code>exprs()</code> Default value NULL
check_is_grouped	Throws an error if dataset is grouped Default value TRUE
check_is_rowwise	Throws an error if dataset is rowwise Default value TRUE
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value <code>rlang::caller_arg(arg)</code>
message	string passed to <code>cli::cli_abort(message)</code> . When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. <code>call = caller_env()</code> . The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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()`, `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()`

Examples

```
library(dplyr)
library(rlang)
dm <- tribble(
  ~STUDYID, ~USUBJID,
  "XYZ",    "1",
  "XYZ",    "2"
)

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"))

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

`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 = rlang::caller_arg(dataset),
```

```

    var_name = rlang::caller_arg(var),
    message = NULL,
    class = "assert_date_var",
    call = parent.frame()
  )

```

Arguments

dataset	The dataset where the variable is expected Default value none
var	The variable to check Default value none
dataset_name	The name of the dataset. If the argument is specified, the specified name is displayed in the error message. Default value rlang::caller_arg(dataset)
var_name	The name of the variable. If the argument is specified, the specified name is displayed in the error message. Default value rlang::caller_arg(var)
message	(string) string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "var_name" and "dataset_name", can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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(lubridate)
library(dplyr)
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,
  arg_name = rlang::caller_arg(arg),
  message = NULL,

```

```

    class = "assert_date_vector",
    call = parent.frame()
)

```

Arguments

arg	The function argument to be checked Default value none
optional	Is the checked argument optional? If set to FALSE and arg is NULL then the function assert_date_vector exits early and throw an error. Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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\(\)](#), [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\(\)](#)

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,
  arg_name = gsub("^enexpr\\((.*)\\)$", "\\1", rlang::caller_arg(arg)),
  message = NULL,
  class = "assert_expr",
  call = parent.frame()
)
```

Arguments

arg	A function argument to be checked Default value none
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value FALSE
arg_name	By default the expression specified for arg is used. If it is of the form enexpr(<argument name>), the enexpr() part is removed. For example if arg = enexpr(filter_add) is specified, arg_name defaults to "filter_add" Default value gsub("^enexpr\\((.*)\\)\$", "\\1", rlang::caller_arg(arg))
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.

`call` The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error.

You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message.

Can also be `NULL` or a [defused function call](#) to respectively not display any call or hard-code a code to display.

For more information about error calls, see [Including function calls in error messages](#).

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()`

<code>assert_expr_list</code>	<i>Is an Argument a List of Expressions?</i>
-------------------------------	--

Description

Checks if the argument is a list of expressions.

Usage

```
assert_expr_list(
  arg,
  required_elements = NULL,
  named = FALSE,
  optional = FALSE,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_expr_list",
  call = parent.frame()
)
```

Arguments

arg	A function argument to be checked Default value none
required_elements	A character vector of names that must be present in arg Default value NULL
named	If set to TRUE, an error is issued if not all elements of the list are named. Default value FALSE
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown. Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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"), !!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,
  arg_name = gsub("^enexpr\\((.*)\\)$", "\\1", rlang::caller_arg(arg)),
  message = NULL,
  class = "assert_filter_cond",
  call = parent.frame()
)
```

Arguments

arg	Quosure - filtering condition. Default value none
optional	Logical - is the argument optional? Default value FALSE
arg_name	By default the expression specified for arg is used. If it is of the form enexpr(<argument name>), the enexpr() part is removed. For example if arg = enexpr(filter_add) is specified, arg_name defaults to "filter_add" Default value gsub("^enexpr\\((.*)\\)\$", "\\1", rlang::caller_arg(arg))
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.

call The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error.

You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message.

Can also be `NULL` or a [defused function call](#) to respectively not display any call or hard-code a code to display.

For more information about error calls, see [Including function calls in error messages](#).

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\(\)](#), [assert_expr_list\(\)](#), [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(dplyr, warn.conflicts = FALSE)
library(rlang)
dm <- dplyr::tribble(
  ~DOMAIN, ~STUDYID, ~USUBJID, ~AGE,
  "DM", "STUDY X", "01-701-1015", 64,
  "DM", "STUDY X", "01-701-1016", 65,
)

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

example_fun(dm, AGE == 64)

try(assert_filter_cond(mtcars))
```

assert_function	<i>Is Argument a Function?</i>
-----------------	--------------------------------

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,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_function",
  call = parent.frame()
)
```

Arguments

arg	A function The function to be checked Default value none
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. Default value NULL
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown. Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.

`call` The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error.

You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message.

Can also be `NULL` or a [defused function call](#) to respectively not display any call or hard-code a code to display.

For more information about error calls, see [Including function calls in error messages](#).

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\(\)](#), [assert_expr_list\(\)](#), [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_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,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_integer_scalar",
  call = parent.frame()
)

```

Arguments

arg	A function argument to be checked Default value none
subset	A subset of integers that arg should be part of. Permitted values "none", "positive", "non-negative", or "negative" Default value "none"
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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\(\)](#), [assert_expr_list\(\)](#), [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,
  arg_name = rlang::caller_arg(list),
  message = NULL,
  class = "assert_list_element",
  call = parent.frame(),
  ...
)
```

Arguments

list	A list to be checked A list of named lists or classes is expected. Default value none
element	The name of an element of the lists/classes A character scalar is expected. Default value none
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. Default value none
message_text	Text to be displayed in the error message above the listing of values that do not meet the condition. The text should describe the condition to be fulfilled, e.g., "Error in {arg_name}: the censor values must be zero.". If message argument is specified, that text will be displayed and message_text is ignored. Default value none
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .
...	Objects required to evaluate the condition or the message text If the condition or the message text contains objects apart from the element, they have to be passed to the function. See the second example below. Default value none

Value

An error if the condition is not met. 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()`, `assert_expr_list()`, `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()`

Examples

```
death <- list(
  dataset_name = "adsl",
  date = "DTHDT",
  censor = 0
)

lstalv <- list(
  dataset_name = "adsl",
  date = "LSTALVDT",
  censor = 1
)

events <- list(death, lstalv)

try(assert_list_element(
  list = events,
  element = "censor",
  condition = censor == 0,
  message_text = "For events the censor values must be zero."
))

try(assert_list_element(
  list = events,
  element = "dataset_name",
  condition = dataset_name %in% c("adrs", "adae"),
  valid_datasets = c("adrs", "adae"),
  message_text = paste(
    "The dataset name must be one of the following: {.val {valid_datasets}}")
  )
))
```

`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,
  cls,
  named = FALSE,
  optional = TRUE,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_list_of",
  call = parent.frame()
)

```

Arguments

arg	A function argument to be checked Default value none
cls	The S3 class or type to check for Default value none
named	If set to TRUE, an error is issued if not all elements of the list are named. Default value FALSE
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value TRUE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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()`, `assert_expr_list()`, `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,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_logical_scalar",
  call = parent.frame()
)
```

Arguments

`arg` A function argument to be checked
Default value none

optional	<p>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.</p> <p>Default value FALSE</p>
arg_name	<p>string indicating the label/symbol of the object being checked.</p> <p>Default value rlang::caller_arg(arg)</p>
message	<p>string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging.</p> <p>Default value NULL</p>
class	<p>Subclass of the condition.</p>
call	<p>The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error.</p> <p>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.</p> <p>Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display.</p> <p>For more information about error calls, see Including function calls in error messages.</p>

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\(\)](#), [assert_expr_list\(\)](#), [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_exist\(\)](#), [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	<i>Assert Argument is a Named List or Vector</i>
--------------	--

Description

Assert that all elements of the argument are named.

Usage

```
assert_named(
  arg,
  optional = FALSE,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_named",
  call = parent.frame()
)
```

Arguments

arg	A function argument to be checked Default value none
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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()`, `assert_expr_list()`, `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_numeric_vector *Is an Argument a Numeric Vector?*

Description

Checks if an argument is a numeric vector

Usage

```
assert_numeric_vector(
  arg,
  length = NULL,
  optional = FALSE,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_numeric_vector",
  call = parent.frame()
)
```

Arguments

arg	A function argument to be checked Default value none
length	Expected length If the argument is not specified or set to NULL, any length is accepted. Default value NULL
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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\(\)](#), [assert_expr_list\(\)](#), [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_exist\(\)](#), [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))

example_fun <- function(num) {
  assert_numeric_vector(num, length = 2)
}

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

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,
  dataset_name = rlang::caller_arg(dataset),
  message = NULL,
  class = "assert_one_to_one",
  call = parent.frame()
)
```

Arguments

dataset	Dataset to be checked The variables specified for vars1 and vars2 are expected. Default value none
vars1	First list of variables Default value none
vars2	Second list of variables Default value none
dataset_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(dataset)

message	string passed to <code>cli::cli_abort(message)</code> . When NULL, default messaging is used (see examples for default messages). "dataset_name" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. <code>call = caller_env()</code> . The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply <code>call</code> when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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()`, `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_param_does_not_exist()`, `assert_s3_class()`, `assert_same_type()`, `assert_symbol()`, `assert_unit()`, `assert_vars()`, `assert_varval_list()`

Examples

```
library(dplyr)
library(rlang)

df <- tribble(
  ~SPECIES, ~SPECIESN,
  "DOG",    1L,
  "CAT",    2L,
  "DOG",    1L
)

assert_one_to_one(df, vars1 = exprs(SPECIES), vars2 = exprs(SPECIESN))

df_many <- tribble(
  ~SPECIES, ~SPECIESN,
  "DOG",    1L,
  "CAT",    2L,
  "DOG",    3L
)
```

```

try(
  assert_one_to_one(df_many, vars1 = exprs(SPECIES), vars2 = exprs(SPECIESN))
)

try(
  assert_one_to_one(df_many, vars1 = exprs(SPECIESN), vars2 = exprs(SPECIES))
)

```

```
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,
  arg_name = rlang::caller_arg(dataset),
  message = NULL,
  class = "assert_param_does_not_exist",
  call = parent.frame()
)

```

Arguments

dataset	A data.frame Default value none
param	Parameter code to check Default value none
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.

Can also be NULL or a [defused function call](#) to respectively not display any call or hard-code a code to display.

For more information about error calls, see [Including function calls in error messages](#).

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\(\)](#), [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_s3_class\(\)](#), [assert_same_type\(\)](#), [assert_symbol\(\)](#), [assert_unit\(\)](#), [assert_vars\(\)](#), [assert_varval_list\(\)](#)

Examples

```
library(dplyr)

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_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,
  cls,
  optional = FALSE,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_s3_class",
  call = parent.frame()
)
```

Arguments

arg	A function argument to be checked Default value none
cls	The S3 class to check for Default value none
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown Default value FALSE
arg_name	string indicating the label/symbol of the object being checked. Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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\(\)](#), [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_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 Arguments of the Same Type?*

Description

Checks if all arguments are of the same type.

Usage

```
assert_same_type(
  ...,
  .message = c("Arguments {.arg {arg_names}} must be the same type.", i =
    paste("Argument types are", paste0("{.arg ", arg_names, "} {.cls ", types, "}",
      collapse = ", "))),
  .class = "assert_same_type",
  .call = parent.frame()
)
```

Arguments

...	Arguments to be checked
	Default value none
.message	character vector passed to cli_abort(message) when assertion fails.
	Default value c("Arguments {.arg {arg_names}} must be the same type.", i = paste("Argument types are", paste0("{.arg ", arg_names, "} {.cls ", types, "}", collapse = ", ")))
.class	character vector passed to cli_abort(class) when assertion fails.
	Default value "assert_same_type"
.call	environment passed to cli_abort(call) when assertion fails.
	Default value parent.frame()

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\(\)](#), [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_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,
  arg_name = gsub("^enexpr\\((.*)\\)", "\\1", rlang::caller_arg(arg)),
  message = NULL,
  class = "assert_symbol",
  call = parent.frame()
)
```

Arguments

arg	A function argument to be checked. Must be a symbol. See examples. Default value none
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown. Default value FALSE
arg_name	By default the expression specified for arg is used. If it is of the form <code>enexpr(<argument name>)</code> , the <code>enexpr()</code> part is removed. For example if <code>arg = enexpr(filter_add)</code> is specified, <code>arg_name</code> defaults to <code>"filter_add"</code> Default value <code>gsub("^enexpr\\((.*)\\)\$", "\\1", rlang::caller_arg(arg))</code>
message	string passed to <code>cli::cli_abort(message)</code> . When NULL, default messaging is used (see examples for default messages). <code>"{arg_name}"</code> can be used in messaging. Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. <code>call = caller_env()</code> . The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply <code>call</code> when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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\(\)](#), [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_unit\(\)](#), [assert_vars\(\)](#), [assert_varval_list\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(rlang)
dm <- dplyr::tribble(
  ~DOMAIN,      ~USUBJID,
  "DM",         "01-701-1015",
```

```

  "DM",    "01-701-1016",
)
example_fun <- function(dat, var) {
  var <- assert_symbol(enexpr(var))
  select(dat, !!var)
}

example_fun(dm, USBJID)

try(example_fun(dm))

try(example_fun(dm, "USBJID"))

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 = NULL,
  get_unit_expr,
  arg_name = rlang::caller_arg(required_unit),
  message = NULL,
  class = "assert_unit",
  call = parent.frame()
)

```

Arguments

dataset	Dataset to be checked The variable PARAMCD and those used in get_unit_expr are expected. Default value none
param	Parameter code of the parameter to check Default value none
required_unit	Expected unit(s) If the argument is set to NULL, it is checked only whether the unit is unique within the parameter. Permitted values A character vector or NULL

	Default value NULL
get_unit_expr	Expression used to provide the unit of param
	Default value none
arg_name	string indicating the label/symbol of the object being checked.
	Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging.
	Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

Value

The function throws an error

- if there is more than one non-missing unit in the dataset or
- if the unit variable differs from the expected 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\(\)](#), [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_vars\(\)](#), [assert_varval_list\(\)](#)

Examples

```
library(dplyr)

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)

try(
  assert_unit(
    advs,
    param = "WEIGHT",
    required_unit = c("g", "mg"),
    get_unit_expr = VSSTRESU
  )
)

# Checking uniqueness of unit only
advs <- tribble(
  ~USUBJID, ~VSTESTCD, ~VSTRESN, ~VSSTRESU, ~PARAMCD, ~AVAL,
  "P01",    "WEIGHT",    80.1, "kg",    "WEIGHT", 80.1,
  "P02",    "WEIGHT",    85700, "g",    "WEIGHT", 85700
)

try(
  assert_unit(advs, param = "WEIGHT", 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,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_vars",
  call = parent.frame()
)

```

Arguments

<code>arg</code>	A function argument to be checked Default value none
<code>expect_names</code>	If the argument is set to TRUE, it is checked if all variables are named, e.g., <code>exprs(APERSDT = APxxSDT, APEREDT = APxxEDT)</code> .

	Default value FALSE
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown
	Default value FALSE
arg_name	string indicating the label/symbol of the object being checked.
	Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging.
	Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

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\(\)](#), [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_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,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_varval_list",
  call = parent.frame()
)

```

Arguments

arg	A function argument to be checked
	Default value none
required_elements	A character vector of names that must be present in arg
	Default value NULL
accept_expr	Should expressions on the right hand side be accepted?
	Default value TRUE
accept_var	Should unnamed variable names (e.g. exprs(USUBJID)) on the right hand side be accepted?

	Default value FALSE
optional	Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown.
	Default value FALSE
arg_name	string indicating the label/symbol of the object being checked.
	Default value rlang::caller_arg(arg)
message	string passed to cli::cli_abort(message). When NULL, default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging.
	Default value NULL
class	Subclass of the condition.
call	The execution environment of a currently running function, e.g. call = caller_env(). The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be NULL or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages .

Value

The function throws an error if arg is not a list of variable-value expressions. 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\(\)](#), [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\(\)](#)

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	<i>Wrap a String in Backquotes</i>
-----------	------------------------------------

Description

Wrap a String in Backquotes

Usage

```
backquote(x)
```

Arguments

x A character vector
Default value none

Value

A character vector

See Also

Helpers for working with Quotes and Quoting: [dquote\(\)](#), [squote\(\)](#)

capture_output	<i>Capture Output and Messages</i>
----------------	------------------------------------

Description

The function captures both output and expected messages from an R expression. If the expression results in an unexpected message, an error is issued.

Usage

```
capture_output(expr, srcref = NULL, expected_cnds = NULL, env = caller_env())
```

Arguments

expr An R expression to evaluate
Permitted values An unquoted R expression
Default value none

srcref The source reference of the expression
Default value NULL

`expected_cnds` A character vector of expected conditions
 If the expression issues a condition of a class that is in this vector, the condition is ignored but added to the return value.
 Otherwise, an error is issued.
Default value `NULL`

`env` The environment in which to evaluate the expression
Default value `caller_env()`

Value

A character vector of captured output and messages

Examples**Capture Output and Messages:**

```
capture_output(1 + 1)
#> [1] "[1] 2"

capture_output(log(-1))
#> Error in capture_output(log(-1)) : The expression
#> > log(-1)
#> issued an unexpected condition:
#> NaNs produced
#> If this is expected, add any of the classes "simpleWarning", "warning", and
#> "condition" to the argument `expected_cnds`.

capture_output(log(-1), expected_cnds = "warning")
#> [1] "[1] NaN"                                "Warning in log(-1) : NaNs produced"
```

`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
Default value `none`

Value

A TRUE if variables were valid variable

See Also

Developer Utility Functions: [%notin%](#), [convert_dtm_to_dtc\(\)](#), [extract_vars\(\)](#), [filter_if\(\)](#), [vars2chr\(\)](#)

convert_dtm_to_dtc	<i>Helper Function to Convert Date (or Date-time) Objects to Characters of dtc Format (-DTC type of variable)</i>
--------------------	---

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
-----	-------------------

Default value none

Value

character vector

See Also

Developer Utility Functions: [%notin%](#), [contains_vars\(\)](#), [extract_vars\(\)](#), [filter_if\(\)](#), [vars2chr\(\)](#)

dataset_vignette	<i>Output a Dataset in a Vignette in the admiral Format</i>
------------------	---

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 Default value none
display_vars	Variables selected to demonstrate the outcome of the derivation 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. Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT) Default value NULL
filter	Filter condition The specified condition is applied to the dataset before it is displayed. Permitted values a condition Default value NULL

Value

A HTML table

deprecate_inform	<i>Deprecation with Soft Message</i>
------------------	--------------------------------------

Description

Wrapper around lifecycle::deprecate_soft() that messages users about deprecated features and functions instead of warning.

Usage

```
deprecate_inform(
  when,
  what,
  with = NULL,
  details = NULL,
  id = NULL,
  env = rlang::caller_env(),
  user_env = rlang::caller_env(2)
)
```

Arguments

when	A string giving the version when the behaviour was deprecated.
what	A string describing what is deprecated: <ul style="list-style-type: none"> • Deprecate a whole function with "foo()". • Deprecate an argument with "foo(arg)". • Partially deprecate an argument with "foo(arg = 'must be a scalar integer')". • Deprecate anything else with a custom message by wrapping it in I(). <p>You can optionally supply the namespace: "ns::foo()", but this is usually not needed as it will be inferred from the caller environment.</p>
with	An optional string giving a recommended replacement for the deprecated behaviour. This takes the same form as what.
details	In most cases the deprecation message can be automatically generated from with. When it can't, use details to provide a hand-written message. details can either be a single string or a character vector, which will be converted to a bulleted list . By default, info bullets are used. Provide a named vectors to override.
id	The id of the deprecation. A warning is issued only once for each id. Defaults to the generated message, but you should provide a unique id when the message in details is built programmatically and depends on inputs, or when you'd like to deprecate multiple functions but warn only once for all of them. Repeated calls to deprecate_soft() and deprecate_warn() are also much faster if you supply an id because it avoids spending time generating the message only to immediately exit if the once per session warning has already been thrown before.
env, user_env	Pair of environments that define where deprecate_*() was called (used to determine the package name) and where the function called the deprecating function was called (used to determine if deprecate_soft() should message). These are only needed if you're calling deprecate_*() from an internal helper, in which case you should forward env = caller_env() and user_env = caller_env(2).

Value

NULL, invisibly.

Examples

```
# A Phase 1 deprecated function with custom bulleted list:
deprecate_inform(
  when = "1.0.0",
  what = "foo()",
  details = c(
    x = "This message will turn into a warning with release of x.y.z",
    i = "See admiral's deprecation guidance:
https://pharmaverse.github.io/admiraldev/dev/articles/programming_strategy.html#deprecation"
  )
)
```

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

Default value none

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\(\)](#), [squote\(\)](#)

enumerate	<i>Enumerate Multiple Elements</i>
-----------	------------------------------------

Description**[Deprecated]**

This function is *deprecated*, please use cli functionality instead.

Usage

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

Arguments

x	A vector or list Default value none
quote_fun	Quoting function, defaults to backquote. If set to NULL, the elements are not quoted. Default value backquote
conjunction	Character to be used in the message, defaults to "and". Default value "and"

Value

A character vector

See Also

Other deprecated: [%or%](#), [arg_name\(\)](#), [friendly_type_of\(\)](#), [valid_time_units\(\)](#), [what_is_it\(\)](#)

expect_dfs_equal	<i>Expectation: Are Two Datasets Equal?</i>
------------------	---

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 Permitted values A dataset, i.e., a data.frame or tibble. Default value none
compare	Comparison dataset Default value none
keys	character vector of variables that define a unique row in the base and compare datasets Default value none
...	Additional arguments passed onto <code>diffdf::diffdf()</code> Default value none

Value

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

Examples

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

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"))

tbl3 <- 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, tbl3, keys = "USUBJID")
```

expr_c	<i>Concatenate One or More Expressions</i>
--------	--

Description

Concatenate One or More Expressions

Usage

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

Arguments

... One or more expressions or list of expressions

Default value none

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	<i>Extract All Symbols from a List of Expressions</i>
--------------	---

Description

Extract All Symbols from a List of Expressions

Usage

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

Arguments

x An R object

Default value none

side One of "lhs" (the default) or "rhs" for formulas

Default value "lhs"

Value

A list of expressions

See Also

Developer Utility Functions: [%notin%](#), [contains_vars\(\)](#), [convert_dtm_to_dtc\(\)](#), [filter_if\(\)](#), [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
	Default value none
filter	A filter condition. Must be an expression.
	Default value none

Value

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

See Also

Developer Utility Functions: [%notin%](#), [contains_vars\(\)](#), [convert_dtm_to_dtc\(\)](#), [extract_vars\(\)](#), [vars2chr\(\)](#)

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

Description

[Deprecated]

This function is *deprecated*, please use `cli` functionality instead.

Usage

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

Arguments

x	Any R object. Default value none
value	Whether to describe the value of x. Default value TRUE
length	Whether to mention the length of vectors and lists. Default value FALSE

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

Other deprecated: `%or%`, `arg_name()`, `enumerate()`, `valid_time_units()`, `what_is_it()`

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. Default value none
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. Default value none
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") Default value NULL

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
Default value none

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
Default value none

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
	Default value none
prefix	The prefix of the new temporary variable name to create
	Default value "tmp_var"

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 dataset 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)
dm <- tribble(
  ~DOMAIN, ~STUDYID, ~USUBJID,
  "DM", "STUDY X", "01-701-1015",
  "DM", "STUDY X", "01-701-1016",
)

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)
```

Arguments

expressions A list of expressions

Default value none

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
Default value none

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_order_vars\(\)](#), [is_valid_dtc\(\)](#)

is_order_vars	<i>Is order vars?</i>
---------------	-----------------------

Description

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

Usage

```
is_order_vars(arg)
```

Arguments

arg An R object
Default value none

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_valid_dtc\(\)](#)

is_valid_dtc	<i>Is this string a valid DTC?</i>
--------------	------------------------------------

Description

Is this string a valid DTC?

Usage

```
is_valid_dtc(arg, check_dtc = FALSE)
```

Arguments

arg	The string to check. Permitted values a character scalar, i.e., a character vector of length one Default value none
check_dtc	If TRUE, the function will check if the string is a real date or datetime (e.g. "2020-02-31" is not real). Note that NA is a valid datetime. Permitted values TRUE, FALSE Default value FALSE

Details

If check_dtc is FALSE, the function only checks if the format of the string is valid, i.e. it will not check if the actual date/datetime is real, so calls such as `is_valid_dtc("2020-13", check_dtc = FALSE)` and `is_valid_dtc("2020-12-01T25:00:00", check_dtc = FALSE)` will return TRUE.

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_order_vars\(\)](#)

Examples

```
## Format is valid, date/datetimes are real
is_valid_dtc("2020-02")
is_valid_dtc("2020-02-28")
is_valid_dtc("2020-02-28T14:43")

# Format is valid, date/datetimes are not real but this is not checked
is_valid_dtc("2020-02-31")
is_valid_dtc("2020-02-28T25:00:00")

# Format is valid, date/datetimes are not real and this is checked
```

```
is_valid_dtc("2020-02-31", check_dtc = TRUE)
is_valid_dtc("2020-02-28T25:00:00", check_dtc = TRUE)
```

parse_code

Parse Code

Description

The function parses the code and returns a list of expressions and source references.

Usage

```
parse_code(code)
```

Arguments

code The code to parse

Permitted values A character vector

Default value none

Value

A list of expressions and source references

Each item of the list is a list with the following elements:

- expr: The expression
- srcref: The source reference
- eval: A logical indicating whether the expression should be evaluated, i.e., it is not a comment or an empty line.

Examples

```
parse_code("1+1\n2*2")
parse_code(c("# sum:", "sum(\n 1, #first\n 2\n)"))
```

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 Default value none
set_values_to	Variables to set A named list returned by <code>exprs()</code> defining the variables to be set, e.g. <code>exprs(PARAMCD = "OS", PARAM = "Overall Survival")</code> is expected. The values must be symbols, character strings, numeric values, expressions, or NA. Default value NULL
expected_types	If the argument is specified, the specified variables are checked whether the specified type matches the type of the variables created by <code>set_values_to</code> . Permitted values A character vector with values "numeric" or "character" Default value NULL

Value

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

Examples

```
library(dplyr)
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")
  )
)

```

rdx_roclet

Roclet Extending the Standard rd Roclet

Description

This roclet extends the standard rd roclet by allowing

- to add permitted values and default values to the @param tag and
- to add a caption and a description to examples.

Usage

```
rdx_roclet()
```

Details

The following tags are supported:

- @permitted: Permitted values for the argument. Permitted value description which are used for several arguments/functions can be stored in inst/roxygen/rdx_meta.R. For example:

```

list(
  rdx_permitted_values = list(
    mode = "`\"first\"`, `\"last\"`,
    msg_type = "`\"none\"`, `\"message\"`, `\"warning\"`, `\"error\"`"
  )
)

```

The reference to the permitted values is done by specifying the name of the list element in square brackets, e.g., @permitted [mode].

- @default: Default value for the argument. By default the default value from the function formals is displayed. This can be overwritten by using the @default tag.
- @examplesx: This tag can be used to mark the beginning of the examples section but doesn't affect the output, i.e., it can be omitted.
- @caption: Caption for the example. The caption is displayed as a subsection in the examples section. The caption can be followed by an arbitrary number of @info and @code tags.
- @info: Description of the example.

- @code: Code of the example.

By default, any warning or error issued by the example code causes the building of the documentation to fail. If this is expected, the condition can be added to the `expected_cnds` option of the `@code` tag. E.g.,

```
@code [expected_cnds = "warning"]
```

To use the roclet call `roxygen2::roxygenise(roclets = "admiral::rdx_roclet")` or add to the DESCRIPTION file:

```
Roxygen: list(markdown = TRUE, roclets = c("collate", "namespace", "admiraldev::rdx_roclet"))
```

For more information on roxygen2 roclets see the [Extending roxygen2](#).

Examples

Using the custom tags:

The id `char_scalar` used for the `@permitted` tag is defined in `man/roxygen/rdx_meta.R`.

See `demo_fun()` for a rendered version of the Rd code generated in the example.

```
roxygen2::roc_proc_text(
  rdx_roclet(),
  c(
    "' A Demo Function",
    "' ",
    "' This function is used to demonstrate the custom tags of the `rdx_roclet`.",
    "' ",
    "' @param x An argument",
    "' @param number A number",
    "' @permitted A number",
    "' @param letter A letter",
    "' @permitted [char_scalar]",
    "' @default The first letter of the alphabet",
    "' @examplesx",
    "' @caption A simple example",
    "' @info This is a simple example showing the default behaviour.",
    "' @code demo_fun(1)",
    "' @caption An example with a different letter",
    "' @info This example shows that the `letter` argument doesn't",
    "'   affect the output. ",
    "' @code demo_fun(1, letter = `b`)",
    "'demo_fun <- function(x, number = 1, letter = `a`) 42"
  )
)
#> $demo_fun.Rd
#> % Generated by roxygen2: do not edit by hand
#> % Please edit documentation in ./<text>
#> \name{demo_fun}
#> \alias{demo_fun}
#> \title{A Demo Function}
```

```

#> \usage{
#> demo_fun(x, number = 1, letter = "a")
#> }
#> \arguments{
#> \item{x}{An argument
#>
#> \describe{
#> \item{Default value}{none}
#> }}
#>
#> \item{number}{A number
#>
#> \describe{
#> \item{Permitted values}{A number}
#> \item{Default value}{\code{1}}
#> }}
#>
#> \item{letter}{A letter
#>
#> \describe{
#> \item{Permitted values}{a character scalar, i.e., a character vector of length one}
#> \item{Default value}{The first letter of the alphabet}
#> }}
#> }
#> \description{
#> This function is used to demonstrate the custom tags of the \code{rdx_roclet()}.
#> }
#> \section{Examples}{
#> \subsection{A simple example}{
#>
#> This is a simple example showing the default behaviour.
#>
#> \if{html}{\out{<div class="sourceCode r">}}\preformatted{demo_fun(1)
#> #> [1] 42}\if{html}{\out{</div>}}
#> \subsection{An example with a different letter}{
#>
#> This example shows that the \code{letter} argument doesn't
#> affect the output.
#>
#> \if{html}{\out{<div class="sourceCode r">}}\preformatted{demo_fun(1, letter = "b")
#> #> [1] 42}\if{html}{\out{</div>}}
#>
#>

```

Description

Remove All Temporary Variables Created Within the Current Function Environment

Usage

```
remove_tmp_vars(dataset)
```

Arguments

dataset The input dataset
Default value none

Value

The input dataset with temporary variables removed

See Also

[get_new_tmp_var\(\)](#)

Examples

```
library(dplyr, warn.conflicts = FALSE)
dm <- tribble(
  ~DOMAIN, ~STUDYID, ~USUBJID,
  "DM", "STUDY X", "01-701-1015",
  "DM", "STUDY X", "01-701-1016",
)
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 Permitted values a quoted expression, e.g., created by <code>expr()</code> Default value none
target	Target symbol Permitted values an unquoted symbol, e.g., AVAL Default value none
replace	Replacing symbol Permitted values an unquoted symbol, e.g., AVAL Default value none

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_symbol_in_expr(expr(if_else(AVAL > 0, AVAL, NA)), AVAL, AVAL.join)
```

`replace_values_by_names`*Replace Expression Value with Name*

Description

Replace Expression Value with Name

Usage

```
replace_values_by_names(expressions)
```

Arguments

`expressions` A list of expressions

Default value none

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)))
```

`roxygen_order_na_handling`*Standard Text for NA Handling in Sorting Variables*

Description

This function provides standardized documentation text about the handling of NA values in sorting variables.

Usage

```
roxygen_order_na_handling()
```

Details

The benefits of having a programmatic way to write documentation is that if any changes need to be made, modifying this function scales across the codebase, can be tested, and is less prone to user-error such as typos or grammar mistakes.

Value

A character string with the standardized documentation text about NA handling in sort order

See Also

Other documentation: [roxygen_param_by_vars\(\)](#), [roxygen_param_dataset\(\)](#), [roxygen_save_memory\(\)](#)

Examples

```
roxygen_order_na_handling()
```

roxygen_param_by_vars *Standard Text for the by_vars Argument*

Description

This function provides standardized documentation text for the `by_vars` argument used in `{admiral}` functions.

Usage

```
roxygen_param_by_vars(rename = FALSE)
```

Arguments

rename	Should the renaming feature be documented? If TRUE, the text includes information about renaming variables using named elements in <code>by_vars</code> .
--------	--

Permitted values TRUE, FALSE

Default value FALSE

Details

The benefits of having a programmatic way to write documentation is that if any changes need to be made, modifying this function scales across the codebase, can be tested, and is less prone to user-error such as typos or grammar mistakes.

Value

A character string with the standardized documentation text for the `by_vars` argument

See Also

Other documentation: [roxygen_order_na_handling\(\)](#), [roxygen_param_dataset\(\)](#), [roxygen_save_memory\(\)](#)

Examples

```
roxygen_param_by_vars()
```

```
roxygen_param_by_vars(rename = TRUE)
```

roxygen_param_dataset *Standard Text for the dataset Argument*

Description

This function provides standardized documentation text for the dataset argument used in {admiral} functions.

Usage

```
roxygen_param_dataset(expected_vars = NULL)
```

Arguments

expected_vars Variables expected in the dataset
A character vector of variable names expected to be present in the dataset argument.

Permitted values A character vector or NULL

Default value NULL

Details

The benefits of having a programmatic way to write documentation is that if any changes need to be made, modifying this function scales across the codebase, can be tested, and is less prone to user-error such as typos or grammar mistakes.

Value

A character string with the standardized documentation text for the dataset argument

See Also

Other documentation: [roxygen_order_na_handling\(\)](#), [roxygen_param_by_vars\(\)](#), [roxygen_save_memory\(\)](#)

Examples

```
roxygen_param_dataset()
```

```
roxygen_param_dataset(expected_vars = c("by_vars"))
```

roxygen_save_memory *Standard Note for Memory-Intensive Functions*

Description

This function provides a standardized note about memory consumption for functions that create large temporary datasets.

Usage

```
roxygen_save_memory()
```

Details

The benefits of having a programmatic way to write documentation is that if any changes need to be made, modifying this function scales across the codebase, can be tested, and is less prone to user-error such as typos or grammar mistakes.

Value

A character string with the standardized note about memory usage

See Also

Other documentation: [roxygen_order_na_handling\(\)](#), [roxygen_param_by_vars\(\)](#), [roxygen_param_dataset\(\)](#)

Examples

```
roxygen_save_memory()
```

squote *Wrap a String in Single Quotes*

Description

Wrap a String in Single Quotes

Usage

```
squote(x)
```

Arguments

x A character vector
Default value none

Value

A character vector

See Also

Helpers for working with Quotes and Quoting: [backquote\(\)](#), [dquote\(\)](#)

suppress_warning	<i>Suppress Specific Warnings</i>
------------------	-----------------------------------

Description

Suppress certain warnings issued by an expression.

Usage

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

Arguments

expr	Expression to be executed Default value none
regexpr	Regular expression matching warnings to suppress Default value none

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	<i>Valid Time Units</i>
------------------	-------------------------

Description**[Deprecated]**

This function is *deprecated*. Please get in touch if you are using this function!

Contains the acceptable character vector of valid time units

Usage

```
valid_time_units()
```

Value

A character vector of valid time units

See Also

Other deprecated: [%or%](#), [arg_name\(\)](#), [enumerate\(\)](#), [friendly_type_of\(\)](#), [what_is_it\(\)](#)

vars2chr	<i>Turn a List of Expressions into a Character Vector</i>
----------	---

Description

Turn a List of Expressions into a Character Vector

Usage

```
vars2chr(expressions)
```

Arguments

expressions A list of expressions created using `exprs()`

Default value none

Value

A character vector

See Also

Developer Utility Functions: [%notin%](#), [contains_vars\(\)](#), [convert_dtm_to_dtc\(\)](#), [extract_vars\(\)](#), [filter_if\(\)](#)

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

Default value none

n A non-negative integer

Default value none

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 Default value none
compare	A named list Default value none
list_name	A string the name of the list Default value none
i	the index id to compare the 2 lists Default value 2

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
	Default value none
is_valid	a logical vector indicating whether elements in dtc are valid
	Default value is_valid_dtc(dtc)

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
	Default value none
vars	character vector of columns to check for in dataset
	Default value none

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(dplyr, warn.conflicts = FALSE)
dm <- tribble(
  ~USUBJID,          ~ARM,
  "01-701-1015", "Placebo",
  "01-701-1016", "Placebo",
)

## 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	<i>What Kind of Object is This?</i>
------------	-------------------------------------

Description**[Deprecated]**

This function is *deprecated*, please use `cli` functionality instead.

Usage

```
what_is_it(x)
```

Arguments

x	Any R object
	Default value none

Value

A character description of the type of x

See Also

Other deprecated: [%or%](#), [arg_name\(\)](#), [enumerate\(\)](#), [friendly_type_of\(\)](#), [valid_time_units\(\)](#)

%notin%	<i>Negated Value Matching</i>
---------	-------------------------------

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
	Default value none
table	The values to be matched against
	Default value none

Value

A logical vector

See Also

Developer Utility Functions: [contains_vars\(\)](#), [convert_dtm_to_dtc\(\)](#), [extract_vars\(\)](#), [filter_if\(\)](#), [vars2chr\(\)](#)

%or%

Or

Description**[Deprecated]**

This function is *deprecated*. Please get in touch if you are using this function!

Usage

```
lhs %or% rhs
```

Arguments

lhs Any valid R expression

Default value none

rhs Any valid R expression

Default value none

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

Other deprecated: [arg_name\(\)](#), [enumerate\(\)](#), [friendly_type_of\(\)](#), [valid_time_units\(\)](#), [what_is_it\(\)](#)

Index

* **assertion**

- assert_atomic_vector, 5
- assert_character_scalar, 7
- assert_character_vector, 9
- assert_data_frame, 10
- assert_date_var, 12
- assert_date_vector, 14
- assert_expr, 16
- assert_expr_list, 17
- assert_filter_cond, 19
- assert_function, 21
- assert_integer_scalar, 22
- assert_list_element, 24
- assert_list_of, 26
- assert_logical_scalar, 28
- assert_named, 30
- assert_numeric_vector, 31
- assert_one_to_one, 33
- assert_param_does_not_exist, 35
- assert_s3_class, 36
- assert_same_type, 38
- assert_symbol, 39
- assert_unit, 41
- assert_vars, 43
- assert_varval_list, 45

* **deprecated**

- %or%, 82
- arg_name, 5
- enumerate, 53
- friendly_type_of, 57
- valid_time_units, 76
- what_is_it, 81

* **dev_utility**

- %notin%, 81
- contains_vars, 48
- convert_dtm_to_dtc, 49
- dataset_vignette, 50
- extract_vars, 55
- filter_if, 56

- vars2chr, 76

* **documentation**

- capture_output, 47
- parse_code, 64
- rdx_roclet, 66
- roxygen_order_na_handling, 71
- roxygen_param_by_vars, 72
- roxygen_param_dataset, 73
- roxygen_save_memory, 74

* **get**

- get_constant_vars, 58
- get_dataset, 58
- get_duplicates, 59
- get_source_vars, 61

* **is**

- is_auto, 61
- is_order_vars, 62
- is_valid_dtc, 63

* **messages**

- deprecate_inform, 50

* **quote**

- backquote, 47
- dquote, 52
- squote, 74

* **quo**

- add_suffix_to_vars, 4
- expr_c, 55
- replace_symbol_in_expr, 70
- replace_values_by_names, 71

* **test_helper**

- expect_dfs_equal, 53

* **tmp_vars**

- get_new_tmp_var, 60
- remove_tmp_vars, 68

* **utils_help**

- process_set_values_to, 65

* **warnings**

- suppress_warning, 75
- warn_if_incomplete_dtc, 77

- warn_if_inconsistent_list, 78
- warn_if_invalid_dtc, 79
- warn_if_vars_exist, 80
- %notin%, 49, 56, 76, 81
- %or%, 5, 53, 57, 76, 81, 82

- add_suffix_to_vars, 4
- add_suffix_to_vars(), 55, 70, 71
- arg_name, 5
- arg_name(), 53, 57, 76, 81, 82
- assert_atomic_vector, 5
- assert_atomic_vector(), 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_character_scalar, 7
- assert_character_scalar(), 6, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_character_vector, 9
- assert_character_vector(), 6, 8, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_data_frame, 10
- assert_data_frame(), 6, 8, 10, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_date_var, 12
- assert_date_vector, 14
- assert_date_vector(), 6, 8, 10, 12, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_expr, 16
- assert_expr(), 6, 8, 10, 12, 15, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_expr_list, 17
- assert_expr_list(), 6, 8, 10, 12, 15, 17, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_filter_cond, 19
- assert_filter_cond(), 6, 8, 10, 12, 15, 17, 18, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_function, 21
- assert_function(), 6, 8, 10, 12, 15, 17, 18, 20, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_integer_scalar, 22
- assert_integer_scalar(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_list_element, 24
- assert_list_element(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 28, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_list_of, 26
- assert_list_of(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 29, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_logical_scalar, 28
- assert_logical_scalar(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 31, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_named, 30
- assert_named(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 32, 34, 36, 37, 39, 40, 42, 44, 46
- assert_numeric_vector, 31
- assert_numeric_vector(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 34, 36, 37, 39, 40, 42, 44, 46
- assert_one_to_one, 33
- assert_one_to_one(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 36, 37, 39, 40, 42, 44, 46
- assert_param_does_not_exist, 35
- assert_param_does_not_exist(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 37, 39, 40, 42, 44, 46
- assert_s3_class, 36
- assert_s3_class(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 39, 40, 42, 44, 46
- assert_same_type, 38
- assert_same_type(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 40, 42, 44, 46
- assert_symbol, 39
- assert_symbol(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 42, 44, 46
- assert_unit, 41
- assert_unit(), 6, 8, 10, 12, 15, 17, 18, 20, 22, 24, 26, 28, 29, 31, 32, 34, 36, 37, 39, 40, 44, 46
- assert_vars, 43

- assert_vars(), [6](#), [8](#), [10](#), [12](#), [15](#), [17](#), [18](#), [20](#), [22](#), [24](#), [26](#), [28](#), [29](#), [31](#), [32](#), [34](#), [36](#), [37](#), [39](#), [40](#), [42](#), [46](#)
- assert_varval_list, [45](#)
- assert_varval_list(), [6](#), [8](#), [10](#), [12](#), [15](#), [17](#), [18](#), [20](#), [22](#), [24](#), [26](#), [28](#), [29](#), [31](#), [32](#), [34](#), [36](#), [37](#), [39](#), [40](#), [42](#), [44](#)
- backquote, [47](#)
- backquote(), [52](#), [75](#)
- bulleted list, [51](#)
- capture_output, [47](#)
- contains_vars, [48](#)
- contains_vars(), [49](#), [56](#), [76](#), [82](#)
- convert_dtm_to_dtc, [49](#)
- convert_dtm_to_dtc(), [49](#), [56](#), [76](#), [82](#)
- dataset_vignette, [50](#)
- defused function call, [6](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#), [18](#), [20](#), [22](#), [23](#), [25](#), [27](#), [29](#), [30](#), [32](#), [34](#), [36](#), [37](#), [40](#), [42](#), [44](#), [46](#)
- demo_fun(), [67](#)
- deprecate_inform, [50](#)
- diffdf::diffdf(), [53](#), [54](#)
- dquote, [52](#)
- dquote(), [47](#), [75](#)
- enumerate, [53](#)
- enumerate(), [5](#), [57](#), [76](#), [81](#), [82](#)
- expect_dfs_equal, [53](#)
- expr_c, [55](#)
- expr_c(), [4](#), [70](#), [71](#)
- extract_vars, [55](#)
- extract_vars(), [49](#), [56](#), [76](#), [82](#)
- filter_if, [56](#)
- filter_if(), [49](#), [56](#), [76](#), [82](#)
- friendly_type_of, [57](#)
- friendly_type_of(), [5](#), [53](#), [76](#), [81](#), [82](#)
- get_constant_vars, [58](#)
- get_constant_vars(), [59–61](#)
- get_dataset, [58](#)
- get_dataset(), [58](#), [60](#), [61](#)
- get_duplicates, [59](#)
- get_duplicates(), [58](#), [59](#), [61](#)
- get_new_tmp_var, [60](#)
- get_new_tmp_var(), [69](#)
- get_source_vars, [61](#)
- get_source_vars(), [58–60](#)
- Including function calls in error messages, [6](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#), [18](#), [20](#), [22](#), [23](#), [25](#), [27](#), [29](#), [30](#), [32](#), [34](#), [36](#), [37](#), [40](#), [42](#), [44](#), [46](#)
- is_auto, [61](#)
- is_auto(), [62](#), [63](#)
- is_order_vars, [62](#)
- is_order_vars(), [62](#), [63](#)
- is_valid_dtc, [63](#)
- is_valid_dtc(), [62](#)
- parse_code, [64](#)
- process_set_values_to, [65](#)
- rdx_roclet, [66](#)
- remove_tmp_vars, [68](#)
- remove_tmp_vars(), [60](#)
- replace_symbol_in_expr, [70](#)
- replace_symbol_in_expr(), [4](#), [55](#), [71](#)
- replace_values_by_names, [71](#)
- replace_values_by_names(), [4](#), [55](#), [70](#)
- roxygen_order_na_handling, [71](#)
- roxygen_order_na_handling(), [73](#), [74](#)
- roxygen_param_by_vars, [72](#)
- roxygen_param_by_vars(), [72–74](#)
- roxygen_param_dataset, [73](#)
- roxygen_param_dataset(), [72–74](#)
- roxygen_save_memory, [74](#)
- roxygen_save_memory(), [72](#), [73](#)
- squote, [74](#)
- squote(), [47](#), [52](#)
- suppress_warning, [75](#)
- suppress_warning(), [77–80](#)
- valid_time_units, [76](#)
- valid_time_units(), [5](#), [53](#), [57](#), [81](#), [82](#)
- vars2chr, [76](#)
- vars2chr(), [49](#), [56](#), [82](#)
- warn_if_incomplete_dtc, [77](#)
- warn_if_incomplete_dtc(), [75](#), [78–80](#)
- warn_if_inconsistent_list, [78](#)
- warn_if_inconsistent_list(), [75](#), [77](#), [79](#), [80](#)
- warn_if_invalid_dtc, [79](#)
- warn_if_invalid_dtc(), [75](#), [77](#), [78](#), [80](#)
- warn_if_vars_exist, [80](#)

`warn_if_vars_exist()`, [75](#), [77–79](#)

`what_is_it`, [81](#)

`what_is_it()`, [5](#), [53](#), [57](#), [76](#), [82](#)