

# Package ‘admiraldev’

June 7, 2024

**Type** Package

**Title** Utility Functions and Development Tools for the Admiral Package Family

**Version** 1.1.0

**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 (>= 4.0)

**Imports** cli, dplyr (>= 1.0.5), glue (>= 1.6.0), lifecycle (>= 0.1.0),  
lubridate (>= 1.7.4), purrr (>= 0.3.3), rlang (>= 0.4.4),  
stringr (>= 1.4.0), tidyr (>= 1.0.2), tidysselect (>= 1.0.0)

**Suggests** diffdf, DT, htmltools, knitr, methods, pkgdown, rmarkdown,  
spelling, testthat (>= 3.2.0), withr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.3.1

**NeedsCompilation** no

**Author** Ben Straub [aut, cre],  
Stefan Bundfuss [aut],  
Jeffrey Dickinson [aut],  
Ross Farrugia [aut],  
Pooja Kumari [aut],

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

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

**Repository** CRAN

**Date/Publication** 2024-06-06 22:10:14 UTC

## Contents

|                                       |    |
|---------------------------------------|----|
| add_suffix_to_vars . . . . .          | 3  |
| anti_join . . . . .                   | 4  |
| arg_name . . . . .                    | 5  |
| assert_atomic_vector . . . . .        | 5  |
| assert_character_scalar . . . . .     | 7  |
| assert_character_vector . . . . .     | 8  |
| assert_data_frame . . . . .           | 10 |
| assert_date_var . . . . .             | 12 |
| assert_date_vector . . . . .          | 13 |
| assert_expr . . . . .                 | 15 |
| assert_expr_list . . . . .            | 16 |
| assert_filter_cond . . . . .          | 17 |
| assert_function . . . . .             | 19 |
| assert_function_param . . . . .       | 21 |
| assert_has_variables . . . . .        | 21 |
| assert_integer_scalar . . . . .       | 22 |
| assert_list_element . . . . .         | 23 |
| assert_list_of . . . . .              | 25 |
| assert_logical_scalar . . . . .       | 26 |
| assert_named . . . . .                | 28 |
| assert_named_exprs . . . . .          | 29 |
| assert_numeric_vector . . . . .       | 30 |
| assert_one_to_one . . . . .           | 31 |
| assert_param_does_not_exist . . . . . | 33 |
| assert_s3_class . . . . .             | 34 |
| assert_same_type . . . . .            | 35 |
| assert_symbol . . . . .               | 37 |
| assert_unit . . . . .                 | 38 |
| assert_vars . . . . .                 | 40 |
| assert_varval_list . . . . .          | 41 |
| backquote . . . . .                   | 43 |

|                                     |    |
|-------------------------------------|----|
| contains_vars . . . . .             | 43 |
| convert_dtm_to_dtc . . . . .        | 44 |
| dataset_vignette . . . . .          | 45 |
| dquote . . . . .                    | 45 |
| enumerate . . . . .                 | 46 |
| expect_dfs_equal . . . . .          | 47 |
| expr_c . . . . .                    | 48 |
| extract_vars . . . . .              | 48 |
| filter_if . . . . .                 | 49 |
| friendly_type_of . . . . .          | 50 |
| get_constant_vars . . . . .         | 50 |
| get_dataset . . . . .               | 51 |
| get_duplicates . . . . .            | 52 |
| get_new_tmp_var . . . . .           | 53 |
| get_source_vars . . . . .           | 54 |
| is_auto . . . . .                   | 54 |
| is_order_vars . . . . .             | 55 |
| is_valid_dtc . . . . .              | 55 |
| process_set_values_to . . . . .     | 56 |
| remove_tmp_vars . . . . .           | 57 |
| replace_symbol_in_expr . . . . .    | 58 |
| replace_values_by_names . . . . .   | 59 |
| squote . . . . .                    | 59 |
| suppress_warning . . . . .          | 60 |
| valid_time_units . . . . .          | 61 |
| vars2chr . . . . .                  | 61 |
| warn_if_incomplete_dtc . . . . .    | 62 |
| warn_if_inconsistent_list . . . . . | 62 |
| warn_if_invalid_dtc . . . . .       | 63 |
| warn_if_vars_exist . . . . .        | 64 |
| what_is_it . . . . .                | 65 |
| %notin% . . . . .                   | 66 |
| %or% . . . . .                      | 66 |

**Index** **68**

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<br><i>Permitted Values:</i> list of variables or desc(<variable>) function calls created by exprs(), e.g., exprs(ADT, desc(AVAL)) |
| vars   | Variables to change<br><i>Permitted Values:</i> list of variables created by exprs()  |
| suffix | Suffix<br><i>Permitted Values:</i> A character scalar   |

**Value**

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

**See Also**

Helpers for working with Quosures: [expr\\_c\(\)](#), [replace\\_symbol\\_in\\_expr\(\)](#), [replace\\_values\\_by\\_names\(\)](#)

**Examples**

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

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

---

anti\_join

*Join Functions*


---

**Description**

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

**Usage**

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

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

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

**Arguments**

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

**Value**

data.frame

---

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

---

**Description**

Extract Argument Name from an Expression

**Usage**

```
arg_name(expr)
```

**Arguments**

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

**Value**

character vector

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [%or%\(\)](#), [contains\\_vars\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [extract\\_vars\(\)](#), [filter\\_if\(\)](#), [friendly\\_type\\_of\(\)](#), [valid\\_time\\_units\(\)](#), [vars2chr\(\)](#)

---

|                      |   |
|----------------------|---|
| assert_atomic_vector | <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   |
| optional | Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown   |
| arg_name | string indicating the label/symbol of the object being checked.   |
| 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.  |
| 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 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 <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.</p> <p>For more information about error calls, see <a href="#">Including function calls in error messages</a>.</p> |

**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  |
| values         | A character vector of valid values for arg. Values is converted to a lower case vector if case_sensitive = FALSE is used.  |
| case_sensitive | Should the argument be handled case-sensitive? If set to FALSE, the argument is converted to lower case for checking the permitted values and returning the argument.  |
| optional       | Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown  |
| arg_name       | string indicating the label/symbol of the object being checked.  |
| 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.   |
| 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.<br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**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**

|          |  |
|----------|--|
| arg      | A function argument to be checked  |
| values   | A character vector of valid values for arg   |
| named    | If set to TRUE, an error is issued if not all elements of the vector are named.  |
| optional | Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown  |
| arg_name | string indicating the label/symbol of the object being checked.  |
| 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.   |
| 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.<br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**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,
  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  |
| required_vars    | A list of variables created using <code>exprs()</code>   |
| check_is_grouped | Throw an error if dataset is grouped? Defaults to TRUE.  |
| optional         | Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown  |
| arg_name         | string indicating the label/symbol of the object being checked.  |
| 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. |

|       |   |
|-------|---|
| 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 <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.</p> <p>For more information about error calls, see <a href="#">Including function calls in error messages</a>.</p> |

### 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   |
| var          | The variable to check  |
| dataset_name | The name of the dataset. If the argument is specified, the specified name is displayed in the error message.   |
| var_name     | The name of the variable. If the argument is specified, the specified name is displayed in the error message.  |
| message      | (string)<br>string passed to <code>cli::cli_abort(message)</code> . When NULL, default messaging is used (see examples for default messages). "var_name" and "dataset_name", can be used in messaging.   |
| class        | Subclass of the condition.   |
| call         | The execution environment of a currently running function, e.g. <code>call = caller_env()</code> .<br>The corresponding function call is retrieved and mentioned in error messages as the source of the error.<br>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.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

### 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  |
| 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.   |
| arg_name | string indicating the label/symbol of the object being checked.  |
| 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.   |
| 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.<br><br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br><br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br><br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**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 | <i>Assert Argument is an Expression</i> |
|-------------|---|

---

**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**

|                       |   |
|-----------------------|---|
| <code>arg</code>      | A function argument to be checked   |
| <code>optional</code> | Is the checked argument optional? If set to FALSE and <code>arg</code> is NULL then an error is thrown  |
| <code>arg_name</code> | By default the expression specified for <code>arg</code> is used. If it is of the form <code>enexpr(&lt;argument name&gt;)</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>   |
| <code>message</code>  | 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.   |
| <code>class</code>    | Subclass of the condition.  |
| <code>call</code>     | 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.<br><br>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 an expression, i.e. either a symbol or a call, or returns the input invisibly otherwise

### See Also

Checks for valid input and returns warning or errors messages: [assert\\_atomic\\_vector\(\)](#), [assert\\_character\\_scalar\(\)](#), [assert\\_character\\_vector\(\)](#), [assert\\_data\\_frame\(\)](#), [assert\\_date\\_vector\(\)](#), [assert\\_expr\\_list\(\)](#), [assert\\_filter\\_cond\(\)](#), [assert\\_function\(\)](#), [assert\\_integer\\_scalar\(\)](#), [assert\\_list\\_element\(\)](#), [assert\\_list\\_of\(\)](#), [assert\\_logical\\_scalar\(\)](#), [assert\\_named\(\)](#), [assert\\_numeric\\_vector\(\)](#), [assert\\_one\\_to\\_one\(\)](#), [assert\\_param\\_does\\_not\\_exist\(\)](#), [assert\\_s3\\_class\(\)](#), [assert\\_same\\_type\(\)](#), [assert\\_symbol\(\)](#), [assert\\_unit\(\)](#), [assert\\_vars\(\)](#), [assert\\_varval\\_list\(\)](#)

---

assert\_expr\_list      *Is an Argument a List of Expressions?*

---

### Description

Checks if the argument is a list of expressions.

### Usage

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

### Arguments

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

|          |   |
|----------|---|
| arg_name | string indicating the label/symbol of the object being checked.   |
| 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.  |
| 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.<br><br>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.<br><br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br><br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

### 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.   |
| optional | Logical - is the argument optional? Defaults to 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"   |
| 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.   |
| 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.<br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**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<br>The function to be checked   |
| params | A character vector<br>A character vector of expected argument names for the aforementioned function in arg. If ellipsis, ..., is included in the function formals of the function in arg, this argument, params will be ignored, accepting all values of the character vector. |

|          |  |
|----------|--|
| optional | Is the checked argument optional?<br>If set to FALSE and arg is NULL then an error is thrown.  |
| arg_name | string indicating the label/symbol of the object being checked.  |
| 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.   |
| class    | Subclass of the condition.   |
| call     | The execution environment of a currently running function, e.g. <code>call = caller_env()</code> .<br>The corresponding function call is retrieved and mentioned in error messages as the source of the error.<br>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.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

## 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\_function\_param *Assert Argument is a Parameter of a Function*

---

### Description

#### [Deprecated]

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

### Usage

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

### Arguments

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

### Value

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

### See Also

Other deprecated: [assert\\_has\\_variables\(\)](#), [assert\\_named\\_exprs\(\)](#)

---

assert\_has\_variables *Does a Dataset Contain All Required Variables?*

---

### Description

#### [Deprecated]

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

### Usage

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

### Arguments

|               |                                      |
|---------------|--------------------------------------|
| dataset       | A <code>data.frame</code>            |
| required_vars | A character vector of variable names |

### Details

Checks if a dataset contains all required variables

**Value**

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

**See Also**

Other deprecated: [assert\\_function\\_param\(\)](#), [assert\\_named\\_exprs\(\)](#)

---

assert\_integer\_scalar *Is an Argument an Integer Scalar?*

---

**Description**

Checks if an argument is an integer scalar

**Usage**

```
assert_integer_scalar(
  arg,
  subset = "none",
  optional = FALSE,
  arg_name = rlang::caller_arg(arg),
  message = NULL,
  class = "assert_integer_scalar",
  call = parent.frame()
)
```

**Arguments**

|          |  |
|----------|--|
| arg      | A function argument to be checked  |
| subset   | A subset of integers that arg should be part of. Should be one of "none" (the default), "positive", "non-negative" or "negative".  |
| optional | Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown  |
| arg_name | string indicating the label/symbol of the object being checked.  |
| 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.   |
| 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.<br>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

|                           |  |
|---------------------------|--|
| <code>list</code>         | A list to be checked A list of named lists or classes is expected.   |
| <code>element</code>      | The name of an element of the lists/classes A character scalar is expected.  |
| <code>condition</code>    | 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., <code>sensor == 0</code> to check the <code>sensor</code> field of a class.  |
| <code>message_text</code> | 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 sensor values must be zero.". If message argument is specified, that text will be displayed and <code>message_text</code> is ignored.  |
| <code>arg_name</code>     | string indicating the label/symbol of the object being checked.  |
| <code>message</code>      | string passed to <code>cli::cli_abort(message)</code> . When <code>NULL</code> , default messaging is used (see examples for default messages). "{arg_name}" can be used in messaging.   |
| <code>class</code>        | Subclass of the condition.   |
| <code>call</code>         | 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.<br>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.<br>Can also be <code>NULL</code> or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |
| <code>...</code>          | Objects required to evaluate the condition If the condition contains objects apart from the element, they have to be passed to the function. See the second example below.   |

## Value

An error if the condition is not 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()

---

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  |
| cls      | The S3 class or type to check for  |
| named    | If set to TRUE, an error is issued if not all elements of the list are named.  |
| optional | Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown  |
| arg_name | string indicating the label/symbol of the object being checked.  |
| 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.   |
| 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.<br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**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   |
| optional | Is the checked argument optional?<br>If set to FALSE and arg is NULL then an error is thrown. Otherwise, NULL is considered as valid value.   |
| arg_name | string indicating the label/symbol of the object being checked.   |
| 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.  |
| class    | Subclass of the condition.  |
| call     | The execution environment of a currently running function, e.g. <code>call = caller_env()</code> .<br>The corresponding function call is retrieved and mentioned in error messages as the source of the error.<br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**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

*Assert Argument is a Named List or Vector*


---

## 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   |
| optional | Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown   |
| arg_name | string indicating the label/symbol of the object being checked.   |
| 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.  |
| 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.<br>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.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

## 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\_named\_exprs      *Assert Argument is a Named List of Expressions*

---

**Description****[Deprecated]**

This function is *deprecated*, please use [assert\\_expr\\_list\(\)](#) instead.

**Usage**

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

**Arguments**

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

**Value**

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

**See Also**

Other deprecated: [assert\\_function\\_param\(\)](#), [assert\\_has\\_variables\(\)](#)

---

**assert\_numeric\_vector** *Is an Argument a Numeric Vector?*

---

**Description**

Checks if an argument is a numeric vector

**Usage**

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

**Arguments**

|                       |   |
|-----------------------|---|
| <code>arg</code>      | A function argument to be checked   |
| <code>optional</code> | Is the checked argument optional? If set to FALSE and <code>arg</code> is NULL then an error is thrown  |
| <code>arg_name</code> | string indicating the label/symbol of the object being checked.   |
| <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.  |
| <code>class</code>    | Subclass of the condition.  |
| <code>call</code>     | 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.<br><br>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.<br><br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br><br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**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))
```

---

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  |
| vars1        | The variables specified for vars1 and vars2 are expected.  |
| vars2        | First list of variables  |
| dataset_name | Second list of variables   |
| message      | string indicating the label/symbol of the object being checked. Default is <code>rlang::caller_arg(dataset)</code>   |
|              | 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. |

**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

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   |
| param    | Parameter code to check  |
| arg_name | string indicating the label/symbol of the object being checked.  |
| 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.   |
| 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.<br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

## 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**

|                       |  |
|-----------------------|--|
| <code>arg</code>      | A function argument to be checked  |
| <code>cls</code>      | The S3 class to check for  |
| <code>optional</code> | Is the checked argument optional? If set to FALSE and <code>arg</code> is NULL then an error is thrown |
| <code>arg_name</code> | string indicating the label/symbol of the object being checked.  |

|         |   |
|---------|---|
| 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.  |
| 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.<br>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.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**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 Argument 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   |
| .message | character vector passed to cli_abort(message) when assertion fails. |
| .class   | character vector passed to cli_abort(class) when assertion fails.   |
| .call    | environment passed to cli_abort(call) when assertion fails.         |

**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 | <i>Is an Argument a Symbol?</i> |
|---------------|---------------------------------|

---

## 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.   |
| optional | Is the checked argument optional? If set to FALSE and arg is NULL then an error is thrown.   |
| 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"   |
| 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.   |
| 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.<br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

## 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, USUBJID)

try(example_fun(dm))

try(example_fun(dm, "USUBJID"))

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

---

 assert\_unit

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


---

**Description**

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

**Usage**

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

**Arguments**

|               |   |
|---------------|---|
| dataset       | A data.frame  |
| param         | Parameter code of the parameter to check  |
| required_unit | Expected unit   |
| get_unit_expr | Expression used to provide the unit of param  |
| arg_name      | string indicating the label/symbol of the object being checked.   |
| 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.  |
| class         | Subclass of the condition.  |
| call          | The execution environment of a currently running function, e.g. call = caller_env().<br>The corresponding function call is retrieved and mentioned in error messages as the source of the error.<br>You only need to supply call when throwing a condition from a helper function which wouldn't be relevant to mention in the message.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

**Value**

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

**See Also**

Checks for valid input and returns warning or errors messages: [assert\\_atomic\\_vector\(\)](#), [assert\\_character\\_scalar\(\)](#), [assert\\_character\\_vector\(\)](#), [assert\\_data\\_frame\(\)](#), [assert\\_date\\_vector\(\)](#), [assert\\_expr\(\)](#), [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)
```

---

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   |
| <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> .  |
| <code>optional</code>     | Is the checked argument optional? If set to FALSE and <code>arg</code> is NULL then an error is thrown  |
| <code>arg_name</code>     | string indicating the label/symbol of the object being checked.   |
| <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.  |
| <code>class</code>        | Subclass of the condition.  |
| <code>call</code>         | 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.<br>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.<br>Can also be NULL or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

## 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

|                                |  |
|--------------------------------|--|
| <code>arg</code>               | A function argument to be checked  |
| <code>required_elements</code> | A character vector of names that must be present in <code>arg</code>   |
| <code>accept_expr</code>       | Should expressions on the right hand side be accepted?   |
| <code>accept_var</code>        | Should unnamed variable names (e.g. <code>exprs(USUBJID)</code> ) on the right hand side be accepted?  |
| <code>optional</code>          | Is the checked argument optional? If set to <code>FALSE</code> and <code>arg</code> is <code>NULL</code> then an error is thrown.  |
| <code>arg_name</code>          | string indicating the label/symbol of the object being checked.  |
| <code>message</code>           | string passed to <code>cli::cli_abort(message)</code> . When <code>NULL</code> , default messaging is used (see examples for default messages). <code>"{arg_name}"</code> can be used in messaging.  |
| <code>class</code>             | Subclass of the condition.   |
| <code>call</code>              | 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.<br><br>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.<br><br>Can also be <code>NULL</code> or a <a href="#">defused function call</a> to respectively not display any call or hard-code a code to display.<br><br>For more information about error calls, see <a href="#">Including function calls in error messages</a> . |

## 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

**Value**

A character vector

**See Also**

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

---

|               |   |
|---------------|---|
| contains_vars | <i>check that argument contains valid variable(s) created with exprs() or Source Variables from a List of Expressions</i> |
|---------------|---|

---

**Description**

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

**Usage**

```
contains_vars(arg)
```

**Arguments**

arg                    A function argument to be checked

**Value**

A TRUE if variables were valid variable

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [%or%\(\)](#), [arg\\_name\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [extract\\_vars\(\)](#), [filter\\_if\(\)](#), [friendly\\_type\\_of\(\)](#), [valid\\_time\\_units\(\)](#), [vars2chr\(\)](#)

---

|                    |   |
|--------------------|---|
| convert_dtm_to_dtc | <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

**Value**

character vector

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [%or%\(\)](#), [arg\\_name\(\)](#), [contains\\_vars\(\)](#), [extract\\_vars\(\)](#), [filter\\_if\(\)](#), [friendly\\_type\\_of\(\)](#), [valid\\_time\\_units\(\)](#), [vars2chr\(\)](#)

---

dataset\_vignette      *Output a Dataset in a Vignette in the admiral Format*

---

### Description

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

### Usage

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

### Arguments

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

### Value

A HTML table

---

dquote      *Wrap a String in Double Quotes*

---

### Description

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

### Usage

```
dquote(x)
```

### Arguments

|   |                    |
|---|--------------------|
| x | A character vector |
|---|--------------------|

**Value**

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

**See Also**

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

---

enumerate

*Enumerate Multiple Elements*

---

**Description**

Enumerate multiple elements of a vector or list.

**Usage**

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

**Arguments**

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

**Value**

A character vector

**See Also**

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

**Examples**

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

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

---

|                  |   |
|------------------|---|
| expect_dfs_equal | <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   |
| compare | Comparison dataset  |
| keys    | character vector of variables that define a unique row in the base and compare datasets |
| ...     | Additional arguments passed onto <code>diffdf::diffdf()</code>                          |

## 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")

```

---

|                     |  |
|---------------------|--|
| <code>expr_c</code> | <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

**Value**

A list of expressions

**See Also**

Helpers for working with Quosures: [add\\_suffix\\_to\\_vars\(\)](#), [replace\\_symbol\\_in\\_expr\(\)](#), [replace\\_values\\_by\\_names\(\)](#)

---

|                           |   |
|---------------------------|---|
| <code>extract_vars</code> | <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  
`side` One of "lhs" (the default) or "rhs" for formulas

**Value**

A list of expressions

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [%or%\(\)](#), [arg\\_name\(\)](#), [contains\\_vars\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [filter\\_if\(\)](#), [friendly\\_type\\_of\(\)](#), [valid\\_time\\_units\(\)](#), [vars2chr\(\)](#)

**Examples**

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

---

filter\_if

*Optional Filter*

---

**Description**

Filters the input dataset if the provided expression is not NULL

**Usage**

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

**Arguments**

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

**Value**

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

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [%or%\(\)](#), [arg\\_name\(\)](#), [contains\\_vars\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [extract\\_vars\(\)](#), [friendly\\_type\\_of\(\)](#), [valid\\_time\\_units\(\)](#), [vars2chr\(\)](#)

---

|                  |   |
|------------------|---|
| friendly_type_of | <i>Return English-friendly messaging for object-types</i> |
|------------------|---|

---

**Description**

Return English-friendly messaging for object-types

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [%or%\(\)](#), [arg\\_name\(\)](#), [contains\\_vars\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [extract\\_vars\(\)](#), [filter\\_if\(\)](#), [valid\\_time\\_units\(\)](#), [vars2chr\(\)](#)

---

|                   |                               |
|-------------------|-------------------------------|
| get_constant_vars | <i>Get Constant Variables</i> |
|-------------------|-------------------------------|

---

**Description**

Get Constant Variables

**Usage**

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

**Arguments**

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

**Value**

Variable vector.

**See Also**

Brings something to you!?: [get\\_dataset\(\)](#), [get\\_duplicates\(\)](#), [get\\_source\\_vars\(\)](#)

---

|             |   |
|-------------|---|
| get_dataset | <i>Retrieve a Dataset from the admiraldev_environment environment</i> |
|-------------|---|

---

**Description**

Retrieve a Dataset from the admiraldev\_environment environment

**Usage**

```
get_dataset(name)
```

**Arguments**

|      |                                     |
|------|-------------------------------------|
| name | The name of the dataset to retrieve |
|------|-------------------------------------|

**Details**

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

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

As so, the establishment of admiraldev\_environment allows us to create dynamic data/objects based on user-inputs that need modification. The purpose of get\_dataset is to retrieve the datasets contained inside admiraldev\_environment.

Currently we only support two datasets inside our admiraldev\_environment object:

- one\_to\_many
- many\_to\_one

**Value**

A data.frame

**See Also**

Brings something to you!?: [get\\_constant\\_vars\(\)](#), [get\\_duplicates\(\)](#), [get\\_source\\_vars\(\)](#)

---

|                |                                     |
|----------------|-------------------------------------|
| get_duplicates | <i>Get Duplicates From a Vector</i> |
|----------------|-------------------------------------|

---

**Description**

Get Duplicates From a Vector

**Usage**

```
get_duplicates(x)
```

**Arguments**

x                    An atomic vector

**Value**

A vector of the same type as x contain duplicate values

**See Also**

Brings something to you!?: [get\\_constant\\_vars\(\)](#), [get\\_dataset\(\)](#), [get\\_source\\_vars\(\)](#)

**Examples**

```
get_duplicates(1:10)
get_duplicates(c("a", "a", "b", "c", "d", "d"))
```

---

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

---

**Description**

Get a New Temporary Variable Name for a Dataset

**Usage**

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

**Arguments**

|                      |   |
|----------------------|---|
| <code>dataset</code> | The input dataset                                       |
| <code>prefix</code>  | The prefix of the new temporary variable name to create |

**Details**

The function returns a new unique temporary variable name to be used inside dataset. The temporary variable names have the structure `prefix_n` where `n` is an integer, e.g. `tmp_var_1`. If there is already a variable inside 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 | <i>Get Source Variables from a List of Expressions</i> |
|-----------------|--|

---

**Description**

Get Source Variables from a List of Expressions

**Usage**

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

**Arguments**

|             |   |
|-------------|---|
| expressions | A list of expressions                               |
| quosures    | <i>Deprecated</i> , please use expressions instead. |

**Value**

A list of expressions

**See Also**

Brings something to you!?: [get\\_constant\\_vars\(\)](#), [get\\_dataset\(\)](#), [get\\_duplicates\(\)](#)

---

|         |   |
|---------|---|
| is_auto | <i>Checks if the argument equals the auto keyword</i> |
|---------|---|

---

**Description**

Checks if the argument equals the auto keyword

**Usage**

```
is_auto(arg)
```

**Arguments**

|     |                   |
|-----|-------------------|
| arg | argument to check |
|-----|-------------------|

**Value**

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

**See Also**

Identifies type of Object with return of TRUE/FALSE: [is\\_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

**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)
```

**Arguments**

arg                    A character vector

**Value**

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

**See Also**

Identifies type of Object with return of TRUE/FALSE: [is\\_auto\(\)](#), [is\\_order\\_vars\(\)](#)

---

process\_set\_values\_to *Process set\_values\_to Argument*

---

### Description

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

### Usage

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

### Arguments

|                             |   |
|-----------------------------|---|
| <code>dataset</code>        | Input dataset   |
| <code>set_values_to</code>  | Variables to set<br>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. |
| <code>expected_types</code> | 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> .<br><i>Permitted Values:</i> A character vector with values "numeric" or "character"              |

### 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")
  )
)
```

---

|                 |   |
|-----------------|---|
| remove_tmp_vars | <i>Remove All Temporary Variables Created Within the Current Function Environment</i> |
|-----------------|---|

---

### Description

Remove All Temporary Variables Created Within the Current Function Environment

### Usage

```
remove_tmp_vars(dataset)
```

### Arguments

dataset            The input dataset

### Value

The input dataset with temporary variables removed

### See Also

[get\\_new\\_tmp\\_var\(\)](#)

### Examples

```
library(dplyr, warn.conflicts = FALSE)
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       |
| target     | Target symbol    |
| replace    | Replacing symbol |

### Value

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

### Author(s)

Stefan Bundfuss

### See Also

Helpers for working with Quosures: [add\\_suffix\\_to\\_vars\(\)](#), [expr\\_c\(\)](#), [replace\\_values\\_by\\_names\(\)](#)

### Examples

```
library(rlang)

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

---

`replace_values_by_names`*Replace Expression Value with Name*

---

**Description**

Replace Expression Value with Name

**Usage**

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

**Arguments**

`expressions`     A list of expressions

`quosures`        *Deprecated*, please use expressions instead.

**Value**

A list of expressions

**See Also**

Helpers for working with Quosures: [add\\_suffix\\_to\\_vars\(\)](#), [expr\\_c\(\)](#), [replace\\_symbol\\_in\\_expr\(\)](#)

**Examples**

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

---

`squote`*Wrap a String in Single Quotes*

---

**Description**

Wrap a String in Single Quotes

**Usage**

```
squote(x)
```

**Arguments**

`x`                A character vector

**Value**

A character vector

**See Also**

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

---

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

---

**Description**

Suppress certain warnings issued by an expression.

**Usage**

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

**Arguments**

|         |  |
|---------|--|
| expr    | Expression to be executed                        |
| regexpr | Regular expression matching warnings to suppress |

**Details**

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

**Value**

Return value of the expression

**See Also**

Function that provide users with custom warnings [warn\\_if\\_incomplete\\_dtc\(\)](#), [warn\\_if\\_inconsistent\\_list\(\)](#), [warn\\_if\\_invalid\\_dtc\(\)](#), [warn\\_if\\_vars\\_exist\(\)](#)

---

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

---

**Description**

Contains the acceptable character vector of valid time units

**Usage**

```
valid_time_units()
```

**Value**

A character vector of valid time units

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [%or%\(\)](#), [arg\\_name\(\)](#), [contains\\_vars\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [extract\\_vars\(\)](#), [filter\\_if\(\)](#), [friendly\\_type\\_of\(\)](#), [vars2chr\(\)](#)

---

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

---

**Description**

Turn a List of Expressions into a Character Vector

**Usage**

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

**Arguments**

|             |   |
|-------------|---|
| expressions | A list of expressions created using <a href="#">exprs()</a> |
| quosures    | <i>Deprecated</i> , please use expressions instead.         |

**Value**

A character vector

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [%or%\(\)](#), [arg\\_name\(\)](#), [contains\\_vars\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [extract\\_vars\(\)](#), [filter\\_if\(\)](#), [friendly\\_type\\_of\(\)](#), [valid\\_time\\_units\(\)](#)

**Examples**

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

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

---

```
warn_if_incomplete_dtc
```

*Warn if incomplete dtc*

---

**Description**

Warn if incomplete dtc

**Usage**

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

**Arguments**

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

**Value**

A warning if dtc contains any partial dates

**See Also**

Function that provide users with custom warnings [suppress\\_warning\(\)](#), [warn\\_if\\_inconsistent\\_list\(\)](#), [warn\\_if\\_invalid\\_dtc\(\)](#), [warn\\_if\\_vars\\_exist\(\)](#)

---

```
warn_if_inconsistent_list
```

*Warn If Two Lists are Inconsistent*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

a warning if the 2 lists have different names or length

**See Also**

Function that provide users with custom warnings [suppress\\_warning\(\)](#), [warn\\_if\\_incomplete\\_dtc\(\)](#), [warn\\_if\\_invalid\\_dtc\(\)](#), [warn\\_if\\_vars\\_exist\(\)](#)

**Examples**

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

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

# warning
warn_if_inconsistent_list(
  base = exprs(DTHDOM = "DM", DTHSEQ = DMSEQ, DTHVAR = "text"),
  compare = exprs(DTHDOM = "DM", DTHSEQ = DMSEQ),
  list_name = "Test"
)
```

---

warn\_if\_invalid\_dtc *Warn If a Vector Contains Unknown Datetime Format*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

No return value, called for side effects

**See Also**

Function that provide users with custom warnings [suppress\\_warning\(\)](#), [warn\\_if\\_incomplete\\_dtc\(\)](#), [warn\\_if\\_inconsistent\\_list\(\)](#), [warn\\_if\\_vars\\_exist\(\)](#)

**Examples**

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

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

---

`warn_if_vars_exist` *Warn If a Variable Already Exists*

---

**Description**

Warn if a variable already exists inside a dataset

**Usage**

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

**Arguments**

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

**Value**

No return value, called for side effects

**See Also**

Function that provide users with custom warnings [suppress\\_warning\(\)](#), [warn\\_if\\_incomplete\\_dtc\(\)](#), [warn\\_if\\_inconsistent\\_list\(\)](#), [warn\\_if\\_invalid\\_dtc\(\)](#)

**Examples**

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

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

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

---

what\_is\_it

*What Kind of Object is This?*

---

**Description**

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

**Usage**

```
what_is_it(x)
```

**Arguments**

x                    Any R object

**Value**

A character description of the type of x

**Examples**

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

---

|         |                               |
|---------|-------------------------------|
| %notin% | <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         |
| table | The values to be matched against |

**Value**

A logical vector

**See Also**

Developer Utility Functions: [%or%\(\)](#), [arg\\_name\(\)](#), [contains\\_vars\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [extract\\_vars\(\)](#), [filter\\_if\(\)](#), [friendly\\_type\\_of\(\)](#), [valid\\_time\\_units\(\)](#), [vars2chr\(\)](#)

---

|      |           |
|------|-----------|
| %or% | <i>Or</i> |
|------|-----------|

---

**Description**

Or

**Usage**

```
lhs %or% rhs
```

**Arguments**

|     |                        |
|-----|------------------------|
| lhs | Any valid R expression |
| rhs | Any valid R expression |

**Details**

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

**Value**

Either the result of evaluating lhs, rhs or an error

**See Also**

Developer Utility Functions: [%notin%\(\)](#), [arg\\_name\(\)](#), [contains\\_vars\(\)](#), [convert\\_dtm\\_to\\_dtc\(\)](#), [extract\\_vars\(\)](#), [filter\\_if\(\)](#), [friendly\\_type\\_of\(\)](#), [valid\\_time\\_units\(\)](#), [vars2chr\(\)](#)

# Index

- \* **assertion**
  - assert\_atomic\_vector, 5
  - assert\_character\_scalar, 7
  - assert\_character\_vector, 8
  - assert\_data\_frame, 10
  - assert\_date\_var, 12
  - assert\_date\_vector, 13
  - assert\_expr, 15
  - assert\_expr\_list, 16
  - assert\_filter\_cond, 17
  - assert\_function, 19
  - assert\_integer\_scalar, 22
  - assert\_list\_element, 23
  - assert\_list\_of, 25
  - assert\_logical\_scalar, 26
  - assert\_named, 28
  - assert\_numeric\_vector, 30
  - assert\_one\_to\_one, 31
  - assert\_param\_does\_not\_exist, 33
  - assert\_s3\_class, 34
  - assert\_same\_type, 35
  - assert\_symbol, 37
  - assert\_unit, 38
  - assert\_vars, 40
  - assert\_varval\_list, 41
- \* **deprecated**
  - assert\_function\_param, 21
  - assert\_has\_variables, 21
  - assert\_named\_exprs, 29
- \* **dev\_utility**
  - %notin%, 66
  - %or%, 66
  - arg\_name, 5
  - contains\_vars, 43
  - convert\_dtm\_to\_dtc, 44
  - dataset\_vignette, 45
  - extract\_vars, 48
  - filter\_if, 49
  - friendly\_type\_of, 50
  - valid\_time\_units, 61
  - vars2chr, 61
- \* **get**
  - get\_constant\_vars, 50
  - get\_dataset, 51
  - get\_duplicates, 52
  - get\_source\_vars, 54
- \* **is**
  - is\_auto, 54
  - is\_order\_vars, 55
  - is\_valid\_dtc, 55
- \* **joins**
  - anti\_join, 4
- \* **quote**
  - backquote, 43
  - dquote, 45
  - enumerate, 46
  - squote, 59
- \* **quo**
  - add\_suffix\_to\_vars, 3
  - expr\_c, 48
  - replace\_symbol\_in\_expr, 58
  - replace\_values\_by\_names, 59
- \* **test\_helper**
  - expect\_dfs\_equal, 47
- \* **tmp\_vars**
  - get\_new\_tmp\_var, 53
  - remove\_tmp\_vars, 57
- \* **utils\_help**
  - process\_set\_values\_to, 56
- \* **warnings**
  - suppress\_warning, 60
  - warn\_if\_incomplete\_dtc, 62
  - warn\_if\_inconsistent\_list, 62
  - warn\_if\_invalid\_dtc, 63
  - warn\_if\_vars\_exist, 64
- \* **what**
  - what\_is\_it, 65
- %notin%, 5, 44, 49, 50, 61, 66, 67

- `%or%`, 5, 44, 49, 50, 61, 66, 66
- `add_suffix_to_vars`, 3, 48, 58, 59
- `anti_join`, 4
- `arg_name`, 5, 44, 49, 50, 61, 66, 67
- `assert_atomic_vector`, 5, 8, 9, 11, 14, 16–18, 20, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_character_scalar`, 6, 7, 9, 11, 14, 16–18, 20, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_character_vector`, 6, 8, 8, 11, 14, 16–18, 20, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_data_frame`, 6, 8, 9, 10, 14, 16–18, 20, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_date_var`, 12
- `assert_date_vector`, 6, 8, 9, 11, 13, 16–18, 20, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_expr`, 6, 8, 9, 11, 14, 15, 17, 18, 20, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_expr_list`, 6, 8, 9, 11, 14, 16, 16, 18, 20, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_filter_cond`, 6, 8, 9, 11, 14, 16, 17, 17, 20, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_function`, 6, 8, 9, 11, 14, 16–18, 19, 23, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_function_param`, 21, 22, 29
- `assert_has_variables`, 21, 21, 29
- `assert_integer_scalar`, 6, 8, 9, 11, 14, 16–18, 20, 22, 24, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_list_element`, 6, 8, 9, 11, 14, 16–18, 20, 23, 23, 26, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_list_of`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25, 25, 27, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_logical_scalar`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25, 26, 26, 29, 31, 32, 34–36, 38, 39, 41, 42
- `assert_named`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 28, 31, 32, 34–36, 38, 39, 41, 42
- `assert_named_exprs`, 21, 22, 29
- `assert_numeric_vector`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 30, 32, 34–36, 38, 39, 41, 42
- `assert_one_to_one`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 31, 31, 34–36, 38, 39, 41, 42
- `assert_param_does_not_exist`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 31, 32, 33, 35, 36, 38, 39, 41, 42
- `assert_s3_class`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 31, 32, 34, 34, 36, 38, 39, 41, 42
- `assert_same_type`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 31, 32, 34, 35, 35, 38, 39, 41, 42
- `assert_symbol`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 31, 32, 34–36, 37, 39, 41, 42
- `assert_unit`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 31, 32, 34–36, 38, 38, 41, 42
- `assert_vars`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 31, 32, 34–36, 38, 39, 40, 42
- `assert_varval_list`, 6, 8, 9, 11, 14, 16–18, 20, 23, 25–27, 29, 31, 32, 34–36, 38, 39, 41, 41
- `backquote`, 43, 46, 60
- `contains_vars`, 5, 43, 44, 49, 50, 61, 66, 67
- `convert_dtm_to_dtc`, 5, 44, 44, 49, 50, 61, 66, 67
- `dataset_vignette`, 45
- `defused function call`, 6, 7, 9, 11, 12, 14, 16–18, 20, 23–25, 27, 28, 30, 32, 33, 35, 37, 39, 40, 42
- `diffdf::diffdf()`, 47
- `dquote`, 43, 45, 46, 60
- `enumerate`, 43, 46, 46, 60
- `expect_dfs_equal`, 47
- `expr_c`, 4, 48, 58, 59
- `exprs()`, 61
- `extract_vars`, 5, 44, 48, 49, 50, 61, 66, 67
- `filter_if`, 5, 44, 49, 49, 50, 61, 66, 67

`friendly_type_of`, [5](#), [44](#), [49](#), [50](#), [61](#), [66](#), [67](#)

`get_constant_vars`, [50](#), [52](#), [54](#)

`get_dataset`, [51](#), [51](#), [52](#), [54](#)

`get_duplicates`, [51](#), [52](#), [52](#), [54](#)

`get_new_tmp_var`, [53](#)

`get_new_tmp_var()`, [57](#)

`get_source_vars`, [51](#), [52](#), [54](#)

Including function calls in error

    messages, [6](#), [7](#), [9](#), [11](#), [12](#), [14](#), [16–18](#),  
    [20](#), [23–25](#), [27](#), [28](#), [30](#), [32](#), [33](#), [35](#), [37](#),  
    [39](#), [40](#), [42](#)

`inner_join(anti_join)`, [4](#)

`is_auto`, [54](#), [55](#)

`is_order_vars`, [54](#), [55](#), [55](#)

`is_valid_dtc`, [54](#), [55](#), [55](#)

`left_join(anti_join)`, [4](#)

`process_set_values_to`, [56](#)

`remove_tmp_vars`, [57](#)

`remove_tmp_vars()`, [53](#)

`replace_symbol_in_expr`, [4](#), [48](#), [58](#), [59](#)

`replace_values_by_names`, [4](#), [48](#), [58](#), [59](#)

`squote`, [43](#), [46](#), [59](#)

`suppress_warning`, [60](#), [62–64](#)

`valid_time_units`, [5](#), [44](#), [49](#), [50](#), [61](#), [61](#), [66](#),  
[67](#)

`vars2chr`, [5](#), [44](#), [49](#), [50](#), [61](#), [61](#), [66](#), [67](#)

`warn_if_incomplete_dtc`, [60](#), [62](#), [63](#), [64](#)

`warn_if_inconsistent_list`, [60](#), [62](#), [62](#), [64](#)

`warn_if_invalid_dtc`, [60](#), [62](#), [63](#), [63](#), [64](#)

`warn_if_vars_exist`, [60](#), [62–64](#), [64](#)

`what_is_it`, [65](#)