

# Package: saferDev (via r-universe)

June 8, 2026

**Type** Package

**Title** Function and Pipeline Development

**Version** 1.0.0

**Date** 2026-06-02

**Encoding** UTF-8

**Maintainer** Gael Millot <gael.millot@pasteur.fr>

**Description** Set of functions that perform checks according to the safer-r project recommendations for R function development (see <<https://github.com/safer-r>>). This includes checking argument values, ensuring correct specification of all mandatory arguments for embedded functions, as well as their explicit package namespace qualification, among other things.

**URL** <<https://github.com/safer-r/saferDev>>,  
<<https://safer-r.github.io/saferDev/>>

**License** GPL-3

**RoxygenNote** 7.3.3

**Imports** ggplot2

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Haiding Wang [ctb], Yushi Han [ctb], Mia Legras [ctb], Gael Millot [cre, aut, ctb] (ORCID:  
<<https://orcid.org/0000-0002-0591-3509>>)

**Repository** <https://cran.r-universe.dev>

**Date/Publication** 2026-06-08 18:10:03 UTC

**RemoteUrl** <https://github.com/cran/saferDev>

**RemoteRef** HEAD

**RemoteSha** 0da70404b693cfa7abb4c7d4cbaf1128f322c2fb

## Contents

all_args_here . . . . .	2
arg_check . . . . .	5
colons_check . . . . .	9
env_check . . . . .	11
get_message . . . . .	14
is_function_here . . . . .	17
is_package_here . . . . .	18
report . . . . .	20

<b>Index</b>	<b>23</b>
--------------	-----------

---

all_args_here	<i>All Arguments Here</i>
---------------	---------------------------

---

### Description

Verify that all the functions used inside a function are written with all their arguments. For instance: `base::paste0(letters[1:2], collapse = NULL, recycle0 = FALSE)` and not `paste0(letters[1:2])`.

### Usage

```
all_args_here(
  x,
  export = FALSE,
  out_path = ".",
  df_name = "res.tsv",
  overwrite = FALSE,
  safer_check = TRUE,
  lib_path = NULL,
  error_text = ""
)
```

### Arguments

x	Function name, written without quotes and brackets.
export	Single logical value. Export the result data frame into a .tsv file? If TRUE, the data frame is not returned by the function but only exported.
out_path	Single character string indicating the absolute pathway of the folder where to export the data frame. <code>out_path = "."</code> means the R working directory set by the user. Ignored if export is FALSE.
df_name	Single character string indicating the name of the exported data frame file. Ignored if export is FALSE.
overwrite	Single logical value. Overwrite potential df_name file already existing in out_path? Ignored if export is FALSE.

safer_check	Single logical value. Perform the "safer" checks? If TRUE, checkings are performed before main code running (see the <a href="#">safer-r project</a> ): 1) correct lib_path argument value 2) required functions and related packages effectively present in local R libraries and 3) R classical operators (like "<-") not overwritten by another package because of the R scope. Warning: must be set to FALSE if this function is used inside another "safer" function to avoid pointless multiple checkings.
lib_path	Vector of characters specifying the absolute pathways of the directories containing the required packages for the function, if not in the default directories. Useful when R packages are not installed in the default directories because of lack of admin rights. More precisely, lib_path is passed through the new argument of .libPaths() so that the new library paths are c(lib_path, .libPaths()). Warning: .libPaths() is restored to the initial paths, after function execution. Ignored if NULL (default) or if the safer_check argument is FALSE: only the pathways specified by the current .libPaths() are used for package calling.
error_text	Single character string used to add information in error messages returned by the function, notably if the function is inside other functions, which is practical for debugging. Example: error_text = " INSIDE <PACKAGE_1>::<FUNCTION_1> INSIDE <PACKAGE_2>::<FUNCTION_2>.". If NULL, converted into "".

## Details

More precisely, all\_args\_here() verifies that all the strings before an opening bracket ( are written with all their arguments. Thus, it cannot check function names written without brackets, like in the FUN argument of some functions, e.g., sapply(1:3, FUN = as.character).

The Perl regex used to detect a function name is: "([a-zA-Z]|\.[a-zA-Z.\_])[a-zA-Z0-9.\_]\*\s\(".

Currently, all\_args\_here() cannot detect functions written between quotes, like "+"() or "rownames<-"(x, "a").

Function names preceded by \$ are not considered.

The following R functions are skipped: function, if, for, while, repeat and else.

Most of the time, all\_args\_here() does not check inside comments, but some unexpected writing could dupe all\_args\_here(). Please, report [here](#) if it is the case.

The returned line numbers are indicative, depending on which source is checked. For instance, saferDev::report (compiled) has not the same line numbers as its [source file](#). Notably, compiled functions do not have comments anymore, compared to the same source function sourced into the working environment. In addition, the counting starts at the "<- function" line, i.e., without counting the #' header lines potentially present in source files.

The function works first by replacing in the code: 1) the three consecutive characters ''' or ''' by three spaces, 2) "\" and \"' by four spaces and 3) escape quotes like \" or \"' by two spaces.

See more examples [here](#).

Warnings:

1. The following R functions are also skipped (as indicated by "SKIPPED" in the DEF\_ARGS column of the returned data frame): [as.environment\(\)](#). Click to see the explanation.

2. Some functions, like `rownames()`, have different arguments depending on whether something is assigned to it (e.g., `rownames(x) <- "a"`) or not. The `all_args_here()` function always proposes the arguments defined in the help page without assignment, meaning that it cannot detect the assignment. A way to bypass this is to use the "exact" writing of the function. For instance, `base::rownames<-"(x, "a)"` instead of using `base::rownames(x) <- "a"`, and use `getAnywhere("rownames<-")` to see the arguments of the function.
3. The function could not properly work if any comma is present in default argument values. Please, report [here](#) if it is the case.
4. Proposals in the STATUS column are only suggestions, as it is difficult to anticipate all the exceptions with arguments writing.

### Value

A data frame indicating the missing arguments or a message saying that everything seems fine. If export argument is TRUE, then the data frame is exported as `res.tsv` instead of being returned. Column descriptions:

- `$LINE_NB`: the line number in the function code (starting at the "`<- function`" line, i.e., without counting the `#'` header lines).
- `$FUN_NAME`: the function name.
- `$FUN_ARGS`: the written arguments of `FUN_NAME`. "NOT\_CONSIDERED" means that the function is between quotes or after `$`.
- `$FUN_POS`: the position of the first character of the function name in the `$LINE_NB` line.
- `$DEF_ARGS`: the defaults arguments of `FUN_NAME`. "NO\_ARGS" means that the function has no arguments. "INTERNAL\_FUNCTION" means that the function has been created inside the checked function. "SKIPPED" means that the function is not analyzed for the reason indicated in the "details" section.
- `$MISSING_ARG_NAMES`: the missing argument names in `FUN_ARGS`.
- `$MISSING_ARGS`: the missing arguments with their values in `FUN_ARGS`.
- `$STATUS`: either "GOOD", meaning that all the arguments are already written, or a new proposal of arguments writing, or indicates if some arguments are not fully written (abbreviation is discouraged), or nothing.

An additional message "EVERYTHING SEEMS CLEAN" if the STATUS column is only made of "" and "GOOD".

### Author(s)

Gael Millot

Haiding Wang

Yushi Han

**Examples**

```

# Warning: these examples may not work well when using the "Run examples" link
# because of a particular environment. Please, copy-paste in a local environment.
# See also https://safer-r.github.io/saferDev/articles/all\_args\_here.html
## Not run:
# Example that returns an error
saferDev::all_args_here(mean) # Example that returns an error
source("https://raw.githubusercontent.com/safer-r/saferDev/main/dev/other/test2.R")
saferDev::all_args_here(test2) # the checked function must be executable

## End(Not run)

FUN2 <- function(x, y){
  middle_bracket2 <- base::do.call(
    what = base::c,
    args = code_for_col,
    quote = FALSE,
    envir = base::parent.frame()
  )
}
saferDev::all_args_here(FUN2, safer_check = FALSE)

## Not run:
# Example that creates a file/folder in the working directory
source("https://raw.githubusercontent.com/safer-r/.github/refs/heads/main/profile/backbone.R")
saferDev::all_args_here(BACKBONE, export = TRUE, safer_check = FALSE)

## End(Not run)

```

---

arg\_check

*Argument Check*


---

**Description**

Check expected values of an argument of functions: class, type, mode, length, restricted values panel, kind of numeric values (in addition to the distinction between 'integer' and 'double')? Proportion only? Inf values authorized? negative values authorized? Integers of type 'double'?

**Usage**

```

arg_check(
  data,
  class = NULL,
  typeof = NULL,
  mode = NULL,
  length = NULL,
  prop = FALSE,
  double_as_integer_allowed = FALSE,

```

```

options = NULL,
all_options_in_data = FALSE,
na_contain = FALSE,
neg_values = TRUE,
inf_values = TRUE,
print = FALSE,
data_name = NULL,
data_arg = TRUE,
safer_check = TRUE,
lib_path = NULL,
error_text = ""
)

```

## Arguments

data	Object to test.
class	Single character string among "vector", "matrix", "array", "data.frame", "list", "factor", "table", "expression", "name", "symbol", "function", "uneval", "environment", "ggplot2", "ggplot_built", "call". simplified version of class(data) (see the details below). Write NULL to do not test the class.
typeof	Single character string among "logical", "integer", "double", "complex", "character", "list", "expression", "symbol", "closure", "special", "builtin", "environment", "S4", "language", "object". Simplified version of checking the type of the tested object using typeof(data). Write NULL to do not test the type.
mode	Single character string among "logical", "numeric", "complex", "character", "list", "expression", "name", "symbol", "function", "environment", "S4", "call", "object". Simplified version of checking the type of the tested object using mode(data). Write NULL to do not test the mode.
length	Single numeric value indicating the length of the object. Not considered if NULL.
prop	Single logical value. Are the numeric values between 0 and 1 (proportion)? If TRUE, can be used alone, i.e., without necessarily checking the class, mode, etc., of the object to test.
double_as_integer_allowed	Single logical value. If TRUE, no error is reported in the checking message if the typeof argument is set to "integer", while the reality is type "double" but with zero as modulo (remainder of a division). This means that <code>i &lt;- 1</code> , which is <code>typeof(i) == "double"</code> is considered as an integer when setting <code>double_as_integer_allowed = TRUE</code> . Warning: <code>double_as_integer_allowed = TRUE</code> uses <code>data %% 1 == 0L</code> but not <code>isTRUE(all.equal(data %% 1, 0))</code> is used here because the argument checks for integers stored as double (does not check for decimal numbers that are approximate integers).
options	Vector of character strings or integers indicating all the possible option values for the data argument, or NULL. Numbers of type "double" are accepted if they have a 0 modulo (i.e., are integer like).

all_options_in_data	Single logical value. If FALSE, the tested object must be made of at least one of the options and nothing else. Example returning no error: data = c(1, 1, 2) and options = c(1, 2, 3). Example returning an error: data = c(1, 4) and options = c(1, 2, 3). If TRUE, the tested object must contain all of the options, at least once, and nothing else. Example returning no error: data = c(1, 1, 2, 3) and options = c(1, 2, 3). Example returning an error (missing 3): data = c(1, 1, 2) and options = c(1, 2, 3). Example returning an error (unwanted 4): data = c(1, 1, 4) and options = c(1, 2, 3). Ignored if the options argument is NULL.
na_contain	Single logical value. Can the data argument contain NA?
neg_values	Single logical value. Are negative numeric values authorized? Warning: the default setting is TRUE, meaning that, in that case, no check is performed for the presence of negative values. The checking is activated only when set to FALSE. In addition, FALSE can only be used when the typeof argument is set to "double" or "integer", or the mode argument to "numeric". Otherwise an error message is returned. Finally, the presence of negative values is not checked with FALSE if the tested object is a factor and a message is returned: "OBJECT MUST BE MADE OF NON NEGATIVE VALUES BUT IS A FACTOR".
inf_values	Single logical value. Are infinite Inf or -Inf values authorized? Warning: the default setting is TRUE, meaning that, in that case, no check is performed for the presence of infinite values. The checking is activated only when set to FALSE. In addition, FALSE can only be used when the typeof argument is set to "double", or the mode argument to "numeric". Otherwise an error message is returned. Finally, the presence of infinite values is not checked with FALSE if the tested object is a factor and a message is returned: "OBJECT MUST BE MADE OF NON NEGATIVE VALUES BUT IS A FACTOR".
print	Single logical value. Print the message if the \$problem output is TRUE? Warning: set by default to FALSE, which facilitates the control of the checking message output when using arg_check() inside functions. See the example section.
data_name	Single character string indicating the name of the object to test. If NULL, use what is assigned to the data argument for the returned message.
data_arg	Single logical value. Is the tested object a function argument? If TRUE (default), "ARGUMENT" is written in output messages, otherwise "OBJECT".
safer_check	Single logical value. Perform some "safer" checks? If TRUE, checkings are performed before main code running (see the <a href="#">safer-r project</a> ): 1) correct lib_path argument value 2) required functions and related packages effectively present in local R libraries and 3) R classical operators (like "<-") not overwritten by another package because of the R scope. Must be set to FALSE if this function is used inside another "safer" function to avoid pointless multiple checkings.
lib_path	Vector of characters specifying the absolute pathways of the directories containing the required packages for the function, if not in the default directories. Useful when R packages are not installed in the default directories because of lack of admin rights. More precisely, lib_path is passed through the new argument of .libPaths() so that the new library paths are unique(c(new, .Library.site, .Library)). Warning: .libPaths() is restored to the initial

paths, after function execution. Ignored if NULL (default) or if the `safer_check` argument is FALSE: only the pathways specified by the current `.libPaths()` are used for package calling.

`error_text` Single character string used to add information in error messages returned by the function, notably if the function is inside other functions, which is practical for debugging. Example: `error_text = " INSIDE <PACKAGE_1>: :<FUNCTION_1> INSIDE <PACKAGE_2>: :<FUNCTION_2>."`. If NULL, converted into `" "`. Of note, in `arg_check()`, `error_text` is also used at the end of the string returned when no problem is detected.

## Details

If `options == NULL`, then at least `class` or `type` or `mode` or `length` argument must be non-null.

If `options` is non-null, then `class`, `type` and `mode` must be NULL, and `length` can be NULL or specified.

The function tests what is written in its arguments, even if what is written is incoherent. For instance, `arg_check(data = factor(1), class = "factor", mode = "character")` will return a problem, whatever the object tested in the `data` argument, because no object can be class `"factor"` and mode `"character"` (factors are class `"factor"` and mode `"numeric"`). Of note, the length of object of class `"environment"` is always 0.

If the tested object is NULL, then the function will always return a checking problem.

The `class` argument is a simplified version of `class(data)`:

- `"vector"` tests objects of class `"numeric"`, `"integer"`, `"character"`, `"logical"`, `"complex"` or `"expression"` (no error if `class(data)` returns one of these values).
- `"matrix"` tests objects of class `"matrix"` for R < 4.0.0 and `c("matrix", "array")` otherwise.
- `"ggplot2"` tests objects of class `c("gg", "ggplot")` for R < 4.5 and `c("ggplot2::ggplot", "ggplot", "ggplot2::gg", "S7_object", "gg")` otherwise.
- `"ggplot2"` tests objects of class `"ggplot_built"` for R < 4.5 and `c("ggplot2::ggplot_built", "ggplot_built", "ggplot2::gg", "S7_object")` otherwise.

Regarding `typeof`, the value `"object"` comes from the fact that if an object has type `OBJSXP` and has the S4 object flag set (`IS_S4_OBJECT(x)` is true), `typeof()` returns `"S4"`. If an object has type `OBJSXP` without the S4 flag set, `typeof()` returns `"object"`.

## Value

A list containing:

- `problem`: logical. Is there any problem detected?
- `text`: message indicating the details of the problem, or the absence of problem.
- `object.name`: value of the `data_name` argument (i.e., name of the checked object if provided, NULL otherwise).

**Author(s)**

Gael Millot

Haiding Wang

Yushi Han

**See Also**

[match.arg.](#)

**Examples**

```
# Warning: these examples may not work well when using the "Run examples" link
# because of a particular environment. Please, copy-paste in a local environment.
# See also https://safer-r.github.io/saferDev/articles/arg\_check.html
test <- matrix(1:3)
## Not run:
# Example that returns an error
saferDev::arg_check(data = test, print = TRUE, class = "vector", mode = "numeric")

## End(Not run)
saferDev::arg_check(data = test, print = TRUE, class = "matrix", mode = "numeric")
saferDev::arg_check(
  data = test,
  print = TRUE,
  class = "matrix",
  mode = "numeric",
  error_text = " using saferDev::arg_check()"
)
```

---

colons\_check

*Colons Check*

---

**Description**

Verify that all the functions used inside a function are preceded by a package attribution. For instance: `base::mean()` and not `mean()`, or `saferDev:::base_op_check()` and not `.base_op_check()`. Warning: does not check that the package is the good one. Use `all_args_here()` for that.

**Usage**

```
colons_check(x, safer_check = TRUE, lib_path = NULL, error_text = "")
```

## Arguments

x	a function name, written without quotes and brackets.
safer_check	Single logical value. Perform some "safer" checks? If TRUE, checkings are performed before main code running (see the <a href="#">safer-r project</a> ): 1) correct lib_path argument value 2) required functions and related packages effectively present in local R libraries and 3) R classical operators (like "<-") not overwritten by another package because of the R scope. Must be set to FALSE if this function is used inside another "safer" function to avoid pointless multiple checkings.
lib_path	Vector of characters specifying the absolute pathways of the directories containing the required packages for the function, if not in the default directories. Useful when R packages are not installed in the default directories because of lack of admin rights. More precisely, lib_path is passed through the new argument of .libPaths() so that the new library paths are unique(c(new, .Library.site, .Library)). Warning: .libPaths() is restored to the initial paths, after function execution. Ignored if NULL (default) or if the safer_check argument is FALSE: only the pathways specified by the current .libPaths() are used for package calling.
error_text	Single character string used to add information in error messages returned by the function, notably if the function is inside other functions, which is practical for debugging. Example: error_text = " INSIDE <PACKAGE_1>::<FUNCTION_1> INSIDE <PACKAGE_2>::<FUNCTION_2>.". If NULL, converted into "".

## Details

The result is used to modify the code inside the tested function like this: <PACKAGE>::<FUNCTION> (or <PACKAGE>:::<FUNCTION> for function names starting by a dot).

More precisely, colons\_check() verifies that all the strings before an opening bracket ( are preceded by ::.

::: are not checked per se, because incorrect writing, like saferDev::.colons\_check\_message() returns an error when executing the tested function, and because base:::sum() is as ok as base::sum(). In the same manner, more than three colons are not checked because it returns an error when executing the tested function.

Warning: colons\_check() cannot check function names written without brackets, like in the FUN argument of some functions, e.g., sapply(1:3, FUN = as.character).

The Perl regex used to detect a function name is: "[a-zA-Z]|\.[a-zA-Z.\_]([a-zA-Z0-9.\_]\*s\*\(".

Currently, colons\_check() cannot detect functions written between quotes, like "+"() or "rownames<-"(x, "a").

Function names preceded by \$ are not considered.

The following R functions are skipped: "function", "if", "for", "while", "repeat" and "else".

Most of the time, colons\_check() does not check inside comments, but some unexpected writing could dupe colons\_check(). Please, report [here](#) if it is the case.

The returned line numbers are indicative, depending on which source is checked. For instance, saferDev::report (compiled) has not the same line numbers as its source file ([source file](#)). Notably, compiled functions do not have comments anymore, compared to the same source function

sourced into the working environment. In addition, the counting starts at the "<- function" line, i.e., without counting the '#' header lines potentially present in source files.

Of note, during package creation, the `devtools::check()` command tells which functions were wrongly attributed to package. Example:

```
checking dependencies in R code ... WARNING
 '::' or ':::' import not declared from: 'sbase'
Missing or unexported objects:
 'base::dev.off' 'base::graphics.off' 'base::hcl' 'base::par' 'base::read.table' 'saferGG::report'
```

### Value

A table-like text message indicating the missing `::` or `:::` or a message saying that everything seems fine.

Table-like: column 1, the line number in the function code (starting at the "<- function" line, i.e., without counting the '#' header lines); column 2, the function name; column 3, the code preceding the function name.

With missing `::` or `:::`, the message also indicates if internal functions are created inside the checked function code, since these functions cannot have `::` or `:::`.

### Author(s)

Gael Millot

Haiding Wang

Yushi Han

### Examples

```
# Warning: these examples may not work well when using the "Run examples" link
# because of a particular environment. Please, copy-paste in a local environment.
# See also https://safer-r.github.io/saferDev/articles/colons\_check.html
saferDev::colons_check(mean)
saferDev::colons_check(colons_check)
source("https://raw.githubusercontent.com/safer-r/saferDev/main/dev/other/test.R")
saferDev::colons_check(test)
```

---

env\_check

*Environment Check*

---

### Description

Verify that object names in the environment defined by the `pos` parameter are identical or not to object names in the above environments (following R Scope). This can be used to verify that names used for objects inside a function or in the working environment do not override names of objects already present in the above R environments, following the R scope.

**Usage**

```
env_check(
  pos = 1,
  name = NULL,
  safer_check = TRUE,
  lib_path = NULL,
  error_text = ""
)
```

**Arguments**

pos	Single non null positive integer indicating the position of the environment checked (argument <code>n</code> of the parent <code>.frame()</code> function). Value 1 means one step above the <code>env_check()</code> local environment (by default). This means that when <code>env_check(pos = 1)</code> is used inside a function <code>A</code> , it checks if the name of any object in the local environment of this function <code>A</code> is also present in above environments, following R Scope, starting by the just above environment. When <code>env_check(pos = 1)</code> is used in the working (Global) environment (named <code>.GlobalEnv</code> ), it checks the object names of this <code>.GlobalEnv</code> environment, in the above environments. See the examples below.
name	Single character string of the name of the checked environment (used in the output string only, where the name of the checked env is replaced by the name value). Can be used when <code>env_check()</code> is used inside a function "A" to specify that the environment of "A" is checked.
safer_check	Single logical value. Perform some "safer" checks? If TRUE, checkings are performed before main code running (see the <a href="#">safer-r project</a> ): 1) correct <code>lib_path</code> argument value 2) required functions and related packages effectively present in local R libraries and 3) R classical operators (like " <code>&lt;-</code> ") not overwritten by another package because of the R scope. Must be set to FALSE if this function is used inside another "safer" function to avoid pointless multiple checkings.
lib_path	Vector of characters specifying the absolute pathways of the directories containing the required packages for the function, if not in the default directories. Useful when R packages are not installed in the default directories because of lack of admin rights. More precisely, <code>lib_path</code> is passed through the new argument of <code>.libPaths()</code> so that the new library paths are <code>unique(c(new, .Library.site, .Library))</code> . Warning: <code>.libPaths()</code> is restored to the initial paths, after function execution. Ignored if NULL (default) or if the <code>safer_check</code> argument is FALSE: only the pathways specified by the current <code>.libPaths()</code> are used for package calling.
error_text	Single character string used to add information in error messages returned by the function, notably if the function is inside other functions, which is practical for debugging. Example: <code>error_text = " INSIDE &lt;PACKAGE_1&gt;: :&lt;FUNCTION_1&gt; INSIDE &lt;PACKAGE_2&gt;: :&lt;FUNCTION_2&gt;."</code> . If NULL, converted into "".

**Value**

A character string indicating the object names of the tested environment that match object names in the above environments, following the R scope, or NULL if no match.

**Author(s)**

Gael Millot

Haiding Wang

Yushi Han

**See Also**[exists.](#)**Examples**

```

# Warning: these examples may not work well when using the "Run examples" link
# because of a particular environment. Please, copy-paste in a local environment.
# See also https://safer-r.github.io/saferDev/articles/env\_check.html
# Examples in the working environment
# creation of the object mean with value 1 in the .GlobalEnv environment,
# knowing that the mean() function also exists in the environment base, above .GlobalEnv:
mean <- 1
# creation of the object t.test with value 1 in the .GlobalEnv environment,
# knowing that the t.test() function also exists in the environment stats, above .GlobalEnv:
t.test <- 1
search() # current R scope (order of the successive R environments).
utils::find("mean") # where the objects with the name "mean" are present.
utils::find("t.test") # where the objects with the name "mean" are present.
# test if any object name of the global environment are above environments:
a <- saferDev::env_check(pos = 1)
a # output string.
cat(a)
# test if any object of the stats environment (one step above .GlobalEnv)
# are in upper environments of stats. Returns NULL since no object names of
# stats are in upper environments:
saferDev::env_check(pos = 2)
rm(mean)
rm(t.test)

# Examples inside a function
# saferDev::env_check() checks if the object names inside the fun1 function
# exist in the .GlobalEnv environment and above:
fun1 <- function() {
  t.test <- 0
  mean <- 5
  saferDev::env_check(pos = 1)
}
a <- fun1()
cat(a) # Warning: cat(fun1()) creates an additional layer of environment.
# saferDev::env_check() checks if the object names inside the environment one step above fun2(),
# here .GlobalEnv, exist in the upper environments of .GlobalEnv:
fun2 <- function() {
  sum <- 0
  saferDev::env_check(pos = 2)
}

```

```

fun2()
# Warning: cat(fun2()) does not return NULL, because the environment tested is not
# anymore .GlobalEnv but inside cat().
a <- fun2()
cat(a) # nothing displayed because fun2() returns NULL
# With the name of the function fun3 indicated in the message:
fun3 <- function() {
  t.test <- 0
  mean <- 5
  saferDev::env_check(pos = 1, name = "fun3")
}
a <- fun3()
cat(a)
# Alternative way:
# sys.calls() gives the name of the imbricated functions and
# sys.calls()[[length(sys.calls())]] the name of the function one step above.
fun4 <- function() {
  t.test <- 0
  mean <- 5
  name <- as.character(sys.calls()[[length(sys.calls())]])
  saferDev::env_check(pos = 1, name = name)
}
a <- fun4()
cat(a)
# A way to have the name of the tested environment according to test.pos value:
fun7 <- function() {
  min <- "VALUE"
  fun8 <- function() {
    test.pos <- 1 # value 1 tests the fun8 env, 2 tests the fun7 env.
    range <- "VALUE"
    name <- if(length(sys.calls()) >= test.pos) {
      as.character(sys.calls()[[length(sys.calls()) + 1 - test.pos]])
    } else {
      search()[[1:length(search())][test.pos - length(sys.calls())]]
    }
    saferDev::env_check(pos = test.pos, name = name)
  }
  fun8()
}
a <- fun7()
cat(a)

```

---

get\_message

*Get Message*


---

## Description

Evaluate an instruction written between "" and return the first of the error message, or the last of the warning or standard (non error non warning) messages if ever exist.

Using argument `print_no = FALSE`, return NULL if nothing reported, which is convenient in some cases.

**Usage**

```

get_message(
  data,
  kind = "error",
  header = TRUE,
  print_no = FALSE,
  text = NULL,
  env = NULL,
  safer_check = TRUE,
  lib_path = NULL,
  error_text = ""
)

```

**Arguments**

data	Single character string of an instruction to evaluate
kind	Single character string. Either "error" to get a potential error message, or "warning" to get a potential warning message, or "message" to get a potential standard (non error and non warning) message.
header	Single logical value. Add a header in the returned message?
print_no	Single logical value. Print a message saying that nothing (NULL output) has to be reported?
text	Single character string added to the header of the output message, even if nothing reported (print_no is TRUE). Ignored if the header argument is FALSE. Write NULL if not required.
env	An object corresponding to an existing environment. Then the data argument value is evaluated only in the indicated environment. Write NULL if not required (R scope is used). Example env = .GlobalEnv. Example env = asNamespace("stats").
safer_check	Single logical value. Perform some "safer" checks? If TRUE, checkings are performed before main code running (see the <a href="#">safer-r project</a> ): 1) correct lib_path argument value 2) required functions and related packages effectively present in local R libraries and 3) R classical operators (like "<-") not overwritten by another package because of the R scope. Must be set to FALSE if this function is used inside another "safer" function to avoid pointless multiple checkings.
lib_path	Vector of characters specifying the absolute pathways of the directories containing the required packages for the function, if not in the default directories. Useful when R packages are not installed in the default directories because of lack of admin rights. More precisely, lib_path is passed through the new argument of .libPaths() so that the new library paths are unique(c(new, .Library.site, .Library)). Warning: .libPaths() is restored to the initial paths, after function execution. Ignored if NULL (default) or if the safer_check argument is FALSE: only the pathways specified by the current .libPaths() are used for package calling.
error_text	Single character string used to add information in error messages returned by the function, notably if the function is inside other functions, which is practical for debugging. Example: error_text = " INSIDE <PACKAGE_1>::<FUNCTION_1> INSIDE <PACKAGE_2>::<FUNCTION_2>.". If NULL, converted into "".

## Details

Warnings:

- Only the first error or last warning/standard message is returned.
- Always use the `env` argument when `get_message()` is used inside functions.
- The function does not prevent printing, for instance if `print()` or `show()` is used inside the instruction tested. To prevent that, use `tempo <- utils::capture.output(error <- get_message(data = "arg_check(data = 'a', class = mean, neg_values = FALSE, print = TRUE)"))`. The return of `get_message()` is assigned into `error` and the printed messages are captured by `utils::capture.output()` and assigned into `tempo`. See the examples.

The `env` argument is used as value for the `envir` argument of the `base::eval()` function.

## Value

The function returns the error, warning, or standard message, as a single character string, depending on what has been selected using the `kind` argument and if such message exists.

NULL if no selected message returned and the `print_no` argument is FALSE.

The following message if no selected message returned and the `print_no` argument is TRUE: "NO (ERROR|WARNING|STANDARD) MESSAGE REPORTED".

The following message if 1) the `kind` argument is "warning" or "message" while an error message is returned and 2) the `print_no` argument is TRUE: "NO POTENTIAL (WARNING|STANDARD) MESSAGE BECAUSE OF ERROR MESSAGE REPORTED".

## Author(s)

Gael Millot

Haiding Wang

Yushi Han

## See Also

[try.](#)

## Examples

```
# Warning: these examples may not work well when using the "Run examples" link
# because of a particular environment. Please, copy-paste in a local environment.
# See also https://safer-r.github.io/saferDev/articles/get\_message.html

# Report error message by default. Here no error to report:
saferDev::get_message(data = "wilcox.test(c(1,1,3), c(1, 2, 4), paired = TRUE)")

# Warning message to report:
saferDev::get_message(data = "wilcox.test(c(1,1,3), c(1, 2, 4), paired = TRUE)", kind = "warning")

# No standard message to report:
saferDev::get_message(data = "wilcox.test(c(1,1,3), c(1, 2, 4), paired = TRUE)", kind = "message",
```

```

print_no = TRUE, text = "IN A")

# Standard message cannot be caught because wilcox.test() returns an error message:
saferDev::get_message(data = "wilcox.test()", kind = "message", print_no = TRUE, text = "IN A")

# Text added in warning message to report:
saferDev::get_message(data = "wilcox.test()", kind = "error", print_no = TRUE, text = "IN A")

# No error message to report with text added:
saferDev::get_message(data = "sum(1)", kind = "error", print_no = TRUE, text = "IN A")

# No error message to report with text added:
saferDev::get_message(data = "message('ahah')", kind = "error", print_no = TRUE, text = "IN A")

# Standard message reported with text added:
saferDev::get_message(data = "message('ahah')", kind = "message", print_no = TRUE, text = "IN A")

# Messages from ggplot2 functions caught:
saferDev::get_message(data = "ggplot2::ggplot(data = data.frame(X = 1:10, stringsAsFactors = TRUE),
mapping = ggplot2::aes(x = X)) + ggplot2::geom_histogram()", kind = "message",
print_no = TRUE, text = "IN INSTRUCTION 1")

```

---

is\_function\_here

*Is Function Here*


---

## Description

Check if required functions are present in installed packages. This controls the potential modifications of package contents.

## Usage

```
is_function_here(fun, safer_check = TRUE, lib_path = NULL, error_text = "")
```

## Arguments

fun	Character vector of the names of the required functions, preceded by the name of the package they belong to and a double or triple colon. Example: <code>c("ggplot2::geom_point", "grid::gpar")</code> . Warning: do not write <code>"()</code> " at the end of each tested function.
safer_check	Single logical value. Perform some "safer" checks? If TRUE, checkings are performed before main code running (see the <a href="#">safer-r project</a> ): 1) correct lib_path argument value 2) required functions and related packages effectively present in local R libraries and 3) R classical operators (like <code>&lt;-</code> ) not overwritten by another package because of the R scope. Must be set to FALSE if this function is used inside another "safer" function to avoid pointless multiple checkings.
lib_path	Vector of characters specifying the absolute pathways of the directories containing the required packages for the function, if not in the default directories. Useful when R packages are not installed in the default directories because

of lack of admin rights. More precisely, `lib_path` is passed through the new argument of `.libPaths()` so that the new library paths are `unique(c(new, .Library.site, .Library))`. Warning: `.libPaths()` is restored to the initial paths, after function execution. Ignored if `NULL` (default) or if the `safer_check` argument is `FALSE`: only the pathways specified by the current `.libPaths()` are used for package calling.

`error_text` Single character string used to add information in error messages returned by the function, notably if the function is inside other functions, which is practical for debugging. Example: `error_text = " INSIDE <PACKAGE_1>::<FUNCTION_1> INSIDE <PACKAGE_2>::<FUNCTION_2>."`. If `NULL`, converted into `" "`.

### Value

An error message if at least one of the checked packages is missing in `lib_path`, or if at least one of the checked functions is missing in the required package, nothing otherwise.

### Author(s)

Gael Millot  
Haiding Wang  
Yushi Han

### See Also

[exists](#) and [findFunction](#)

### Examples

```
# Warning: these examples may not work well when using the "Run examples" link
# because of a particular environment. Please, copy-paste in a local environment.
# See also https://safer-r.github.io/saferDev/articles/is\_function\_here.html
## Not run:
# Example that returns an error
saferDev::is_function_here(fun = "ggplot2::notgood", error_text = " INSIDE P1::F1")
saferDev::is_function_here(fun = c("ggplot2::geom_point", "grid::gpar"))
saferDev::is_function_here(fun = "c")

## End(Not run)
saferDev::is_function_here(fun = "base::c", error_text = " INSIDE P1::F1")
```

---

is\_package\_here

*Is Package Here*

---

### Description

Check if required packages are installed locally.

**Usage**

```
is_package_here(  
  req_package,  
  safer_check = TRUE,  
  lib_path = NULL,  
  error_text = ""  
)
```

**Arguments**

req_package	Character vector of package names to check.
safer_check	Single logical value. Perform some "safer" checks? If TRUE, checkings are performed before main code running (see the <a href="#">safer-r project</a> ): 1) correct lib_path argument value 2) required functions and related packages effectively present in local R libraries and 3) R classical operators (like "<-") not overwritten by another package because of the R scope. Must be set to FALSE if this function is used inside another "safer" function to avoid pointless multiple checkings.
lib_path	Vector of characters specifying the absolute pathways of the directories containing the required packages for the function, if not in the default directories. Useful when R packages are not installed in the default directories because of lack of admin rights. More precisely, lib_path is passed through the new argument of .libPaths() so that the new library paths are unique(c(lib_path, .libPaths())). Warning: .libPaths() is restored to the initial paths, after function execution. Ignored if NULL (default): only the pathways specified by the current .libPaths() are used for package calling.
error_text	Single character string used to add information in error messages returned by the function, notably if the function is inside other functions, which is practical for debugging. Example: error_text = " INSIDE <PACKAGE_1>: :<FUNCTION_1> INSIDE <PACKAGE_2>: :<FUNCTION_2>.". If NULL, converted into "".

**Value**

An error message if at least one of the checked packages is missing in lib\_path, nothing otherwise.

**Author(s)**

Gael Millot

Haiding Wang

Yushi Han

**See Also**

[require.](#)

## Examples

```
# Warning: these examples may not work well when using the "Run examples" link
# because of a particular environment. Please, copy-paste in a local environment.
# See also https://safer-r.github.io/saferDev/articles/is\_package\_here.html
## Not run:
# Example that returns an error
is_package_here(req_package = "nopackage", error_text = " INSIDE P1::F1")
# Example that returns an error
is_package_here(req_package = "ggplot2", lib_path = "NOTGOOD")

## End(Not run)
is_package_here(req_package = "ggplot2")
```

---

report

*Report*

---

## Description

Print a character string or a data object into a same output file. Convenient for log file generation.

## Usage

```
report(
  data,
  output = "log.txt",
  path,
  append = FALSE,
  rownames_kept = FALSE,
  vector_cat = FALSE,
  noquote = TRUE,
  sep = 2,
  safer_check = TRUE,
  lib_path = NULL,
  error_text = ""
)
```

## Arguments

data	Object to print in the output file. If NULL, nothing is done, with no warning.
output	Single character string. Name of the output file.
path	Single character string indicating the path where to write the output file.
append	Single logical value. If the output file already exists and append is FALSE, an error message is returned (no overwrite of existing file). Otherwise, the printing is appended (and the output file is created if it does not exist yet).

rownames_kept	Single logical value. Defines whether row names have to be removed or not in 2D objects. Warning: in 1D tables, names over the values are taken as row names, and are thus removed if rownames_kept is FALSE.
vector_cat	Single logical value. If TRUE, print a vector of length > 1 using <code>cat()</code> instead of <code>capture.output()</code> . Otherwise (default FALSE) print a vector of length > 1 using <code>capture.output()</code> . Names of values are not printed when TRUE.
noquote	Single logical value. If TRUE no quote are added to the returned character strings.
sep	Single non null and positive integer representing the number of empty lines added in the output file after the printed data.
safer_check	Single logical value. Perform some "safer" checks? If TRUE, checkings are performed before main code running (see the <a href="#">safer-r project</a> ): 1) correct <code>lib_path</code> argument value 2) required functions and related packages effectively present in local R libraries and 3) R classical operators (like "<-") not overwritten by another package because of the R scope. Must be set to FALSE if this function is used inside another "safer" function to avoid pointless multiple checkings.
lib_path	Vector of characters specifying the absolute pathways of the directories containing the required packages for the function, if not in the default directories. Useful when R packages are not installed in the default directories because of lack of admin rights. More precisely, <code>lib_path</code> is passed through the new argument of <code>.libPaths()</code> so that the new library paths are <code>unique(c(new, .Library.site, .Library))</code> . Warning: <code>.libPaths()</code> is restored to the initial paths, after function execution. Ignored if NULL (default) or if the <code>safer_check</code> argument is FALSE: only the pathways specified by the current <code>.libPaths()</code> are used for package calling.
error_text	Single character string used to add information in error messages returned by the function, notably if the function is inside other functions, which is practical for debugging. Example: <code>error_text = " INSIDE &lt;PACKAGE_1&gt;: :&lt;FUNCTION_1&gt; INSIDE &lt;PACKAGE_2&gt;: :&lt;FUNCTION_2&gt;."</code> . If NULL, converted into "".

## Value

Nothing (implementation of an existing file) or a text file.

## Author(s)

[Gael Millot](#)

Haiding Wang

Yushi Han

## See Also

[capture.output](#).

## Examples

```
# Warning: these examples may not work well when using the "Run examples" link
# because of a particular environment. Please, copy-paste in a local environment.
```

```
# See also https://safer-r.github.io/saferDev/articles/report.html
## Not run:
# Example that creates a file/folder in the working directory
saferDev::report(data = "THE FOLLOWING VECTOR IS:\n", output = "results.txt", path = ".",
append = TRUE, sep = 1)
saferDev::report(data = 1:3, output = "results.txt", path = ".", append = TRUE,
rownames_kept = FALSE, vector_cat = FALSE, noquote = FALSE, sep = 2)
saferDev::report(data = "THE FOLLOWING MATRIX IS:\n", output = "results.txt", path = ".",
append = TRUE, sep = 1)
saferDev::report(data = matrix(1:5), output = "results.txt", path = ".", append = TRUE,
rownames_kept = FALSE, vector_cat = FALSE, noquote = FALSE, sep = 5)
saferDev::report(data = "THE FOLLOWING DATA FRAME IS:\n", output = "results.txt", path = ".",
append = TRUE, sep = 1)
saferDev::report(data = data.frame(A = 1:8, B = letters[1:8]), output = "results.txt", path = ".",
append = TRUE, rownames_kept = FALSE, vector_cat = FALSE, noquote = FALSE, sep = 1)

## End(Not run)
```

# Index

`all_args_here`, [2](#)  
`arg_check`, [5](#)

`capture.output`, [21](#)  
`colons_check`, [9](#)

`env_check`, [11](#)  
`exists`, [13](#), [18](#)

`findFunction`, [18](#)

`get_message`, [14](#)

`is_function_here`, [17](#)  
`is_package_here`, [18](#)

`match.arg`, [9](#)

`report`, [20](#)  
`require`, [19](#)

`try`, [16](#)