Package 'rtemis.core'

April 21, 2026

Title Core Utilities for the 'rtemis' Ecosystem

Version 0.0.3

Date 2026-04-19

Description Utilities used across packages of the 'rtemis' ecosystem. Includes the msg() messaging system and the fmt() formatting system. Provides test_* functions that return logical values, check_* functions that throw informative errors, and clean_* functions that return validated and coerced values. This code began as part of the 'rtemis' package (<doi:10.32614/CRAN.package.rtemis>).

License GPL (>= 3)

URL https://www.rtemis.org, https://github.com/rtemis-org/rtemis.core

BugReports https://github.com/rtemis-org/rtemis.core/issues

Depends R (>= 4.1.0)

Encoding UTF-8

RoxygenNote 7.3.3

Imports cli, data.table, methods, S7

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author E.D. Gennatas [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-9280-3609>)

Maintainer E.D. Gennatas <gennatas@gmail.com>

Repository CRAN

Date/Publication 2026-04-21 19:52:51 UTC

Contents

rtemis.core-package 3
abbreviate_class 3

ansi256_to_hex 4
bold 4
check_character 5
check_data.table 6
check_dependencies 6
check_enum 7
check_float01exc 8
check_float01inc 8
check_float0pos 9
check_floatpos 10
check_floatpos1 11
check_float_neg1_1 11
check_inherits 12
check_logical 13
check_optional_scalar_character 14
check_scalar_logical 14
check_tabular 15
clean_colnames 16
clean_int 16
clean_names 17
clean_posint 18
col256 19
ddSci 19
fmt 20
fmt_gradient 22
get_output_type 23
highlight 23
labelify 24
match_arg 25
msg 26
msgdone 27
msgstart 28
optional 28
plain 29
printdf 30
printls 31
repr 32
repr_ls 33
rtemis_colors 35
test_inherits 35

Index **37**

rtemis.core-package **rtemis.core**: *Rtemis Utilities*

Description

Core Utilities for rtemis R Packages

Author(s)

Maintainer: E.D. Gennatas <gennatas@gmail.com> (ORCID) [copyright holder]

See Also

Useful links:

- https://www.rtemis.org
- https://github.com/rtemis-org/rtemis.core
- Report bugs at https://github.com/rtemis-org/rtemis.core/issues

abbreviate_class *Abbreviate object class name*

Description

Abbreviate object class name

Usage

```
abbreviate_class(x, n = 4L)
```

Arguments

x	Object.
n	Integer: Minimum abbreviation length.

Value

Character: Abbreviated class wrapped in angle brackets.

Author(s)

EDG

Examples

```
abbreviate_class(iris)
abbreviate_class(iris, n = 3)
```

ansi256_to_hex *Convert ANSI 256 color code to HEX*

Description

Convert ANSI 256 color code to HEX

Usage

```
ansi256_to_hex(code)
```

Arguments

code	Integer: ANSI 256 color code (0-255).

Value

Character: HEX color string.

Author(s)

EDG

Examples

```
ansi256_to_hex(1)
```

bold *Make text bold*

Description

A `fmt()` convenience wrapper for making text bold.

Usage

```
bold(text, output_type = c("ansi", "html", "plain"))
```

Arguments

text	Character: Text to make bold
output_type	Character: Output type ("ansi", "html", "plain")

Value

Character: Formatted text with bold styling

Author(s)

EDG

Examples

```
message(bold("This is bold!"))
```

---

check_character *Check character*

Description

Check character

Usage

```
check_character(x, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

x	Vector to check.
allow_null	Logical: If TRUE, NULL values are allowed and return early.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if check fails.

Author(s)

EDG

Examples

```
check_character("papaya")
# Throws error:
try(check_character(42L))
```

check_data.table *Check data.table*

Description

Check data.table

Usage

```
check_data.table(x, arg_name = deparse(substitute(x)))
```

Arguments

x	Object to check.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if input is not a data.table, returns x invisibly otherwise.

Author(s)

EDG

Examples

```
check_data.table(data.table::as.data.table(iris))
# Throws error:
try(check_data.table(iris))
```

check_dependencies **rtemis.core** *internal: Dependencies check*

Description

Checks if dependencies can be loaded; names missing dependencies if not.

Usage

```
check_dependencies(..., verbosity = 0L)
```

Arguments

...	List or vector of strings defining namespaces to be checked
verbosity	Integer: Verbosity level. Note: An error will always printed if dependencies are missing. Setting this to FALSE stops it from printing "Dependencies check passed".

Value

Called for side effects. Aborts and prints list of missing dependencies, if any.

Author(s)

EDG

Examples

```
check_dependencies("base")
# Throws error:
try(check_dependencies("zlorbglorb"))
```

---

check_enum *Check if value is in set of allowed values*

Description

Checks if a value is in a set of allowed values, and throws an error if not.

Usage

```
check_enum(x, allowed_values, arg_name = deparse(substitute(x)))
```

Arguments

x	Value to check.
allowed_values	Vector of allowed values.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if x is not in allowed_values, returns x invisibly otherwise.

Author(s)

EDG

Examples

```
check_enum("apple", c("apple", "banana", "cherry"))
# Throws error:
try(check_enum("granola", c("croissant", "bagel", "scramble")))
```

check_float01exc *Check float between 0 and 1, exclusive*

Description

Check float between 0 and 1, exclusive

Usage

```
check_float01exc(x, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

x	Numeric vector.
allow_null	Logical: If TRUE, NULL values are allowed and return early.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if checks fail.

Author(s)

EDG

Examples

```
check_float01exc(c(0.2, 0.7))
# Throws error:
try(check_float01exc(c(0, 0.5, 1)))
```

check_float01inc *Check float between 0 and 1, inclusive*

Description

Check float between 0 and 1, inclusive

Usage

```
check_float01inc(x, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

x	Numeric vector.
allow_null	Logical: If TRUE, NULL values are allowed and return early.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if checks fail.

Author(s)

EDG

Examples

```
check_float01inc(0.5)
```

`check_float0pos` *Check float greater than or equal to 0*

Description

Checks if an input is a numeric vector containing non-negative (>= 0) values and no NAs. It is designed to validate function arguments.

Usage

```
check_float0pos(x, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

x	Numeric vector.
allow_null	Logical: If TRUE, NULL values are allowed and return early.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if checks fail.

Author(s)

EDG

Examples

```
check_float0pos(c(0, 0.5, 1))
# Allows integers since they are numeric and can be coerced to double without loss of information
check_float0pos(c(0L, 1L))
# Throws error:
try(check_float0pos(c(-1.5, 0, 1.5)))
```

check_floatpos *Check positive float*

Description

Check positive float

Usage

```
check_floatpos(x, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

x	Numeric vector.
allow_null	Logical: If TRUE, NULL values are allowed and return early.
arg_name	Character: Name of the variable for error messages.

Details

Checking with `is.numeric()` allows integer inputs as well, which should be ok since it is unlikely the function that consumes this will enforce double type only, but instead is most likely to allow implicit coercion from integer to numeric.

Value

Called for side effects. Throws an error if checks fail.

Author(s)

EDG

Examples

```
check_floatpos(c(0.5, 1.5))
# Allows integers since they are numeric and can be coerced to double without loss of information
check_floatpos(c(1L, 3L))
# Throws error:
try(check_floatpos(c(-1.5, 0.5, 1.5)))
```

check_floatpos1 *Check float in (0, 1]*

Description

Check float in (0, 1]

Usage

```
check_floatpos1(x, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

x	Numeric vector.
allow_null	Logical: If TRUE, NULL values are allowed and return early.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if checks fail.

Author(s)

EDG

Examples

```
check_floatpos1(c(0.5, 1))
# Throw error:
try(check_floatpos1(c(0, 0.7)))
try(check_floatpos1(c(0.5, 1.5)))
```

check_float_neg1_1 *Check float -1 <= x <= 1*

Description

Check float -1 <= x <= 1

Usage

```
check_float_neg1_1(x, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

x	Numeric vector.
allow_null	Logical: If TRUE, NULL values are allowed and return early.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if checks fail.

Author(s)

EDG

Examples

```
check_float_neg1_1(c(-1, 0, 1))
# Throws error:
try(check_float_neg1_1(c(-1.5, 0, 1.5)))
```

---

check_inherits *Check class of object*

Description

Check class of object

Usage

```
check_inherits(x, cl, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

x	Object to check.
cl	Character: class to check against.
allow_null	Logical: If TRUE, NULL values are allowed and return early.
arg_name	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if checks fail.

Author(s)

EDG

Examples

```
check_inherits("papaya", "character")
# These will throw errors:
try(check_inherits(c(1, 2.5, 3.2), "integer"))
try(check_inherits(iris, "list"))
```

`check_logical` *Check logical*

Description

Check logical

Usage

```
check_logical(x, allow_null = TRUE, arg_name = deparse(substitute(x)))
```

Arguments

`x`	Vector to check.
`allow_null`	Logical: If TRUE, NULL values are allowed and return early.
`arg_name`	Character: Name of the variable for error messages.

Value

Called for side effects. Throws an error if checks fail.

Author(s)

EDG

Examples

```
check_logical(c(TRUE, FALSE))
# Throws error:
try(check_logical(c(0, 1)))
```

check_optional_scalar_character *Check Optional Scalar Character*

Description

Check Optional Scalar Character

Usage

```
check_optional_scalar_character(x, arg_name)
```

Arguments

x	Optional Character: Value to check.
arg_name	Character: Argument name to use in error messages.

Value

Called for side effects.

Author(s)

EDG

Examples

```
check_optional_scalar_character(NULL, "my_arg") # Passes
check_optional_scalar_character("hello", "my_arg") # Passes
# Throw error:
try(check_optional_scalar_character(c("hello", "world"), "my_arg"))
try(check_optional_scalar_character(123, "my_arg"))
```

check_scalar_logical *Check Scalar Logical*

Description

Check Scalar Logical

Usage

```
check_scalar_logical(x, arg_name)
```

Arguments

x	Logical: Value to check.
arg_name	Character: Argument name to use in error messages.

Value

Called for side effects.

Author(s)

EDG

Examples

```
check_scalar_logical(TRUE, "my_arg") # Passes
check_scalar_logical(FALSE, "my_arg") # Passes
# Throw error:
try(check_scalar_logical(c(TRUE, FALSE), "my_arg"))
try(check_scalar_logical(NA, "my_arg"))
try(check_scalar_logical("TRUE", "my_arg"))
```

---

check_tabular *Check object is tabular*

Description

Checks if object is of class `data.frame`, `data.table`, or `tbl_df`.

Usage

```
check_tabular(x)
```

Arguments

x	Object to check.

Value

Called for side effects. Throws an error if input is not tabular, returns x invisibly otherwise.

Author(s)

EDG

Examples

```
check_tabular(iris)
check_tabular(data.table::as.data.table(iris))
# Throws error:
try(check_tabular(matrix(1:10, ncol = 2)))
```

clean_colnames *Clean column names*

Description

Clean column names by replacing all spaces and punctuation with a single underscore

Usage

```
clean_colnames(x)
```

Arguments

x	Character vector or matrix with colnames or any object with `names()` method.

Value

Character vector.

Author(s)

EDG

Examples

```
clean_colnames(iris)
```

clean_int *Clean integer input*

Description

Clean integer input

Usage

```
clean_int(x, xname = deparse(substitute(x)))
```

Arguments

x	Double or integer vector to check.
xname	Character: Name of the variable for error messages.

Details

The goal is to return an integer vector. If the input is integer, it is returned as is. If the input is numeric, it is coerced to integer only if the numeric values are integers, otherwise an error is thrown.

Value

Integer vector.

Author(s)

EDG

Examples

```
clean_int(6L)
clean_int(3)
# clean_int(12.1) # Error
clean_int(c(3, 5, 7))
# clean_int(c(3, 5, 7.01)) # Error
```

---

clean_names *Clean names*

Description

Clean character vector by replacing all symbols and sequences of symbols with single underscores, ensuring no name begins or ends with a symbol

Usage

```
clean_names(x, prefix_digits = "V_")
```

Arguments

x	Character vector.
prefix_digits	Character: prefix to add to names beginning with a digit. Set to NA to skip.

Value

Character vector.

Author(s)

EDG

Examples

```
x <- c("Patient ID", "_Date-of-Birth", "SBP (mmHg)")
x
clean_names(x)
```

`clean_posint` *Check positive integer*

Description

Check positive integer

Usage

```
clean_posint(x, allow_na = FALSE, xname = deparse(substitute(x)))
```

Arguments

`x`	Integer vector.
`allow_na`	Logical: If TRUE, NAs are excluded before checking. If FALSE (default), NAs trigger an error.
`xname`	Character: Name of the variable for error messages.

Value

Integer vector of positive values.

Author(s)

EDG

Examples

```
clean_posint(5)
```

col256 *Apply 256-color formatting*

Description

Apply 256-color formatting

Usage

```
col256(text, col = "79", bg = FALSE, output_type = c("ansi", "html", "plain"))
```

Arguments

text	Character: Text to color
col	Character or numeric: Color (ANSI 256-color code, hex for HTML)
bg	Logical: If TRUE, apply as background color
output_type	Character: Output type ("ansi", "html", "plain")

Value

Character: Formatted text with 256-color styling

Author(s)

EDG

Examples

```
col256("Hello", col = 160, output_type = "ansi")
```

ddSci *Format Numbers for Printing*

Description

2 Decimal places, otherwise scientific notation

Usage

```
ddSci(x, decimal_places = 2, hi = 1e+06, as_numeric = FALSE)
```

Arguments

x	Vector of numbers
decimal_places	Integer: Return this many decimal places.
hi	Float: Threshold at or above which scientific notation is used.
as_numeric	Logical: If TRUE, convert to numeric before returning. This will not force all numbers to print 2 decimal places. For example: 1.2035 becomes "1.20" if as_numeric = FALSE, but 1.2 otherwise This can be helpful if you want to be able to use the output as numbers / not just for printing.

Details

Numbers will be formatted to 2 decimal places, unless this results in 0.00 (e.g. if input was .0032), in which case they will be converted to scientific notation with 2 significant figures. ddSci will return 0.00 if the input is exactly zero. This function can be used to format numbers in plots, on the console, in logs, etc.

Value

Formatted number

Author(s)

EDG

Examples

```
x <- .34876549
ddSci(x)
# "0.35"
x <- .00000000457823
ddSci(x)
# "4.6e-09"
```

---

fmt *Text formatting*

Description

Formats text with specified color, styles, and background using ANSI escape codes or HTML, with support for plain text output.

Usage

```
fmt(
  x,
  col = NULL,
  bold = FALSE,
  italic = FALSE,
  underline = FALSE,
  thin = FALSE,
  muted = FALSE,
  bg = NULL,
  pad = 0L,
  output_type = c("ansi", "html", "plain")
)
```

Arguments

x	Character: Text to format.
col	Character: Color (hex code, named color, or NULL for no color).
bold	Logical: If TRUE, make text bold.
italic	Logical: If TRUE, make text italic.
underline	Logical: If TRUE, underline text.
thin	Logical: If TRUE, make text thin/light.
muted	Logical: If TRUE, make text muted/dimmed.
bg	Character: Background color (hex code, named color, or NULL).
pad	Integer: Number of spaces to pad before text.
output_type	Character: Output type ("ansi", "html", "plain").

Details

This function combines multiple formatting options into a single call, making it more efficient than nested function calls. It generates optimized ANSI escape sequences and clean HTML output.

Value

Character: Formatted text with specified styling.

Author(s)

EDG

Examples

```
# Simple color
fmt("Hello", col = "red")

# Bold red text
fmt("Error", col = "red", bold = TRUE)
```

```
# Multiple styles
fmt("Warning", col = "yellow", bold = TRUE, italic = TRUE)

# With background
fmt("Highlight", col = "white", bg = "blue", bold = TRUE)
```

`fmt_gradient` *Gradient text*

Description

Gradient text

Usage

```
fmt_gradient(x, colors, bold = FALSE, output_type = c("ansi", "html", "plain"))
```

Arguments

`x`	Character: Text to colorize.
`colors`	Character vector: Colors to use for the gradient.
`bold`	Logical: If TRUE, make text bold.
`output_type`	Character: Output type ("ansi", "html", "plain").

Value

Character: Text with gradient color applied.

Author(s)

EDG

Examples

```
fmt_gradient("Gradient Text", colors = c("blue", "red")) |> message()
```

get_output_type *Get output type*

Description

Get output type for printing text.

Usage

```
get_output_type(output_type = c("ansi", "html", "plain"), filename = NULL)
```

Arguments

output_type	Character vector of output types.
filename	Optional Character: Filename for output.

Details

Exported as internal function for use by other rtemis packages.

Value

Character with selected output type.

Author(s)

EDG

Examples

```
get_output_type()
```

highlight *Highlight text*

Description

A `fmt()` convenience wrapper for highlighting text.

Usage

```
highlight(x, pad = 0L, output_type = c("ansi", "html", "plain"))
```

Arguments

x	Character: Text to highlight.
pad	Integer: Number of spaces to pad before text.
output_type	Character: Output type ("ansi", "html", "plain").

Value

Character: Formatted text with highlight.

Author(s)

EDG

Examples

```
message(highlight("This is highlighted!"))
```

---

labelify *Format text for label printing*

Description

Format text for label printing

Usage

```
labelify(
  x,
  underscores_to_spaces = TRUE,
  dotsToSpaces = TRUE,
  toLower = FALSE,
  toTitleCase = TRUE,
  capitalize_strings = c("id"),
  stringsToSpaces = c("\\$", "`")
)
```

Arguments

x	Character: Input
underscores_to_spaces	Logical: If TRUE, convert underscores to spaces.
dotsToSpaces	Logical: If TRUE, convert dots to spaces.
toLower	Logical: If TRUE, convert to lowercase (precedes toTitleCase). Default = FALSE (Good for getting all-caps words converted to title case, bad for abbreviations you want to keep all-caps)

toTitleCase	Logical: If TRUE, convert to Title Case. Default = TRUE (This does not change all-caps words, set toLower to TRUE if desired)
capitalize_strings	Character, vector: Always capitalize these strings, if present. Default = "id"
stringsToSpaces	Character, vector: Replace these strings with spaces. Escape as needed for gsub. Default = "\\$", which formats common input of the type data.frame$variable

Value

Character vector.

Author(s)

EDG

Examples

```
x <- c("county_name", "total.cost$", "age", "weight.kg")
labelify(x)
```

---

match_arg *Match Arguments Ignoring Case*

Description

Match Arguments Ignoring Case

Usage

```
match_arg(x, choices)
```

Arguments

x	Character: Argument to match.
choices	Character vector: Choices to match against.

Value

Character: Matched argument.

Author(s)

EDG

Examples

```
match_arg("papaya", c("AppleExtreme", "SuperBanana", "PapayaMaster"))
```

`msg` *Message with provenance*

Description

Print message to output with a prefix including data and time, and calling function or full call stack

Usage

```
msg(
  ...,
  caller = NULL,
  call_depth = 1L,
  caller_id = 1L,
  newline_pre = FALSE,
  newline = TRUE,
  format_fn = plain,
  sep = " ",
  verbosity = 1L
)

msg0(
  ...,
  caller = NULL,
  call_depth = 1,
  caller_id = 1,
  newline_pre = FALSE,
  newline = TRUE,
  format_fn = plain,
  sep = "",
  verbosity = 1L
)
```

Arguments

`...`	Message to print
`caller`	Character: Name of calling function
`call_depth`	Integer: Print the system call path of this depth.
`caller_id`	Integer: Which function in the call stack to print
`newline_pre`	Logical: If TRUE begin with a new line.
`newline`	Logical: If TRUE end with a new line.
`format_fn`	Function: Formatting function to use on the message text.
`sep`	Character: Use to separate objects in `...`
`verbosity`	Integer: Verbosity level of the message. If 0L, does not print anything and returns NULL, invisibly.

Details

If `msg` is called directly from the console, it will print `[interactive>]` in place of the call stack. `msg0`, similar to `paste0`, is `msg(..., sep = "")`

Value

If verbosity > 0L, returns a list with call, message, and date, invisibly, otherwise returns NULL invisibly.

Author(s)

EDG

Examples

```
msg("Hello")
```

---

msgdone *msgdone*

Description

msgdone

Usage

```
msgdone(caller = NULL, call_depth = 1, caller_id = 1, sep = " ")
```

Arguments

`caller`	Character: Name of calling function
`call_depth`	Integer: Print the system call path of this depth.
`caller_id`	Integer: Which function in the call stack to print
`sep`	Character: Use to separate objects in `...`

Value

NULL invisibly

Author(s)

EDG

Examples

```
msgstart("Starting process...")
msgdone("Process complete")
```

msgstart *msgstart*

Description

msgstart

Usage

```
msgstart(..., newline_pre = FALSE, sep = "")
```

Arguments

...	Message to print
newline_pre	Logical: If TRUE begin with a new line.
sep	Character: Use to separate objects in ...

Value

NULL invisibly

Author(s)

EDG

Examples

```
msgstart("Starting process...")
msgdone("Process complete.")
```

optional *Create an optional S7 type*

Description

Creates an S7 union type that allows for the specified type or NULL.

Usage

```
optional(type)
```

Arguments

type	S7 base class or S7 class.

Value

An S7 union type that allows for the specified type or NULL.

Author(s)

EDG

Examples

```
# Create an optional character type
optional(S7::class_character)
```

---

plain *Force plain text when using* message()

Description

Force plain text when using message()

Usage

```
plain(x)
```

Arguments

x	Character: Text to be output to console.

Value

Character: Text with ANSI escape codes removed.

Author(s)

EDG

Examples

```
message(plain("hello"))
```

printdf *Print data frame*

Description

Pretty print a data frame

Usage

```
printdf(
  x,
  pad = 0,
  spacing = 1,
  ddSci_dp = NULL,
  transpose = FALSE,
  justify = "right",
  colnames = TRUE,
  rownames = TRUE,
  column_fmt = highlight,
  row_fmt = gray,
  newline_pre = FALSE,
  newline = FALSE
)
```

Arguments

x	data frame
pad	Integer: Pad output with this many spaces.
spacing	Integer: Number of spaces between columns.
ddSci_dp	Integer: Number of decimal places to print using ddSci. Default = NULL for no formatting.
transpose	Logical: If TRUE, transpose x before printing.
justify	Character: "right", "left".
colnames	Logical: If TRUE, print column names.
rownames	Logical: If TRUE, print row names.
column_fmt	Formatting fn for printing column names.
row_fmt	Formatting fn for printing row names.
newline_pre	Logical: If TRUE, print a new line before printing data frame.
newline	Logical: If TRUE, print a new line after printing data frame.

Details

By design, numbers will not be justified, but using ddSci_dp will convert to characters, which will be justified. This is intentional for internal use.

Value

NULL invisibly

Author(s)

EDG

Examples

```
printdf(iris[1:6, ])
```

`printls` *Pretty print list*

Description

Pretty print a list (or data frame) recursively

Usage

```
printls(
  x,
  prefix = "",
  pad = 2L,
  item_format = bold,
  maxlength = 4L,
  center_title = TRUE,
  title = NULL,
  title_newline = TRUE,
  newline_pre = FALSE,
  format_fn_rhs = ddSci,
  print_class = TRUE,
  abbrev_class_n = 3L,
  print_df = FALSE,
  print_S4 = FALSE,
  limit = 12L
)
```

Arguments

`x`	list or object that will be converted to a list.
`prefix`	Character: Optional prefix for names.
`pad`	Integer: Pad output with this many spaces.
`item_format`	Formatting function for list item names.
`maxlength`	Integer: Maximum length of items to show using `headdot()` before truncating with ellipsis.

center_title	Logical: If TRUE, autopad title for centering, if present.
title	Character: Optional title to print before list.
title_newline	Logical: If TRUE, print title on new line.
newline_pre	Logical: If TRUE, print newline before list.
format_fn_rhs	Formatting function for right-hand side values.
print_class	Logical: If TRUE, print abbreviated class of object.
abbrev_class_n	Integer: Number of characters to abbreviate class names to.
print_df	Logical: If TRUE, print data frame contents, otherwise print n rows and columns.
print_S4	Logical: If TRUE, print S4 object contents, otherwise print class name.
limit	Integer: Maximum number of items to show. Use -1 for unlimited.

Details

Data frames in R began life as lists

Value

NULL invisibly

Author(s)

EDG

Examples

```
printls(list(a = 1:10, b = "Hello", c = list(d = 1, e = 2)), title = "A List")
```

---

repr *String representation*

Description

String representation

Usage

```
repr(x, ...)
```

Arguments

x	Object to represent as a string.
...	Additional arguments passed to methods.

Value

Character string representation of the object.

Author(s)

EDG

Examples

```
S7::method(repr, S7::class_character) <- function(x, ...) {
  paste0("<chr> \"", x, "\"")
}
cat(repr("hello"))
```

---

repr_ls *Show list as formatted string*

Description

Works exactly like printls, but instead of printing to console with cat, it outputs a single string, formatted using mformat, so that cat(repr_ls(x)) looks identical to printls(x) for any list x

Usage

```
repr_ls(
  x,
  prefix = "",
  pad = 2L,
  item_format = bold,
  maxlength = 4L,
  center_title = TRUE,
  title = NULL,
  title_newline = TRUE,
  newline_pre = FALSE,
  format_fn_rhs = ddSci,
  print_class = TRUE,
  abbrev_class_n = 3L,
  print_df = FALSE,
  print_S4 = FALSE,
  limit = 12L,
  output_type = c("ansi", "html", "plain")
)
```

Arguments

x	list or object that will be converted to a list.
prefix	Character: Optional prefix for names.
pad	Integer: Pad output with this many spaces.
item_format	Formatting function for items.

maxlength	Integer: Maximum length of items to show using headdot() before truncating with ellipsis.
center_title	Logical: If TRUE, autopad title for centering, if present.
title	Character: Title to print before list.
title_newline	Logical: If TRUE, print title on new line.
newline_pre	Logical: If TRUE, print newline before list.
format_fn_rhs	Formatting function for right-hand side of items.
print_class	Logical: If TRUE, print abbreviated class of object.
abbrev_class_n	Integer: Number of characters to abbreviate class names to.
print_df	Logical: If TRUE, print data frame contents, otherwise print n rows and columns.
print_S4	Logical: If TRUE, print S4 object contents, otherwise print class name.
limit	Integer: Maximum number of items to show.
output_type	Character: Output type for mformat ("ansi", "html", "plain").

Details

Exported as internal function for use by other rtemis packages.

Value

Character: Formatted string that can be printed with cat()

Author(s)

EDG

Examples

```
x <- list(
  a = 1:10,
  b = "Hello",
  c = list(
    d = 1,
    e = 2
  )
)
cat(repr_ls(x, title = "A List"))
```

rtemis_colors *rtemis Colors*

Description

A named vector of colors used in the rtemis ecosystem, provided as hex strings.

Usage

```
rtemis_colors
```

Format

An object of class `character` of length 12.

Value

Named character vector of hex color codes.

Author(s)

EDG

Examples

```
rtemis_colors[["teal"]]
```

test_inherits *Check class of object*

Description

Check class of object

Usage

```
test_inherits(x, cl)
```

Arguments

`x`	Object to check
`cl`	Character: class to check against

Value

Logical

Author(s)

EDG

Examples

```
test_inherits("papaya", "character") # TRUE
test_inherits(c(1, 2.5, 3.2), "integer")
test_inherits(iris, "list") # FALSE, compare to is_check(iris, is.list)
```

Index

∗ **datasets**
    rtemis_colors, 35

abbreviate_class, 3
ansi256_to_hex, 4

bold, 4

check_character, 5
check_data.table, 6
check_dependencies, 6
check_enum, 7
check_float01exc, 8
check_float01inc, 8
check_float0pos, 9
check_float_neg1_1, 11
check_floatpos, 10
check_floatpos1, 11
check_inherits, 12
check_logical, 13
check_optional_scalar_character, 14
check_scalar_logical, 14
check_tabular, 15
clean_colnames, 16
clean_int, 16
clean_names, 17
clean_posint, 18
col256, 19

ddSci, 19, *30*

fmt, 20
fmt_gradient, 22

get_output_type, 23

highlight, 23

labelify, 24

match_arg, 25
msg, 26
msg0 *(*msg*)*, 26
msgdone, 27
msgstart, 28

optional, 28

plain, 29
printdf, 30
printls, 31

repr, 32
repr_ls, 33
rtemis.core *(*rtemis.core-package*)*, 3
rtemis.core-package, 3
rtemis_colors, 35

test_inherits, 35