

# Package ‘rtemis.core’

June 30, 2026

**Title** Core Utilities for the 'rtemis' Ecosystem

**Version** 0.3.1

**Date** 2026-06-30

**Description** Utilities used across packages of the 'rtemis' ecosystem. Includes the `msg()` messaging system and the `fmt()` formatting system. Provides a library of 'S7' properties, `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](https://doi.org/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

**Imports** data.table, methods, S7

**Suggests** testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

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

**Maintainer** E.D. Gennatas <[gennatas@gmail.com](mailto:gennatas@gmail.com)>

**Repository** CRAN

**Date/Publication** 2026-06-30 08:40:02 UTC

## Contents

rtemis.core-package . . . . .	4
abbreviate_class . . . . .	5

abort . . . . .	5
ansi256_to_hex . . . . .	7
bold . . . . .	7
bounded_double_property . . . . .	8
character_scalar . . . . .	9
check_character . . . . .	9
check_character_scalar . . . . .	10
check_data.table . . . . .	11
check_dependencies . . . . .	11
check_double_scalar . . . . .	12
check_enum . . . . .	13
check_file_exists . . . . .	13
check_float0lexc . . . . .	14
check_float0line . . . . .	15
check_float0pos . . . . .	15
check_floatpos . . . . .	16
check_floatpos1 . . . . .	17
check_float_neg1_1 . . . . .	18
check_inherits . . . . .	18
check_integer_scalar . . . . .	19
check_logical . . . . .	20
check_logical_scalar . . . . .	21
check_nonneg_double_scalar . . . . .	21
check_nonneg_double_vector . . . . .	22
check_numeric . . . . .	23
check_optional_character_scalar . . . . .	24
check_optional_double_scalar . . . . .	24
check_optional_integer_scalar . . . . .	25
check_optional_logical_scalar . . . . .	26
check_optional_nonneg_double_scalar . . . . .	27
check_optional_nonneg_double_vector . . . . .	27
check_optional_pos_double_scalar . . . . .	28
check_optional_pos_double_vector . . . . .	29
check_optional_pos_integer_scalar . . . . .	30
check_optional_prob_scalar . . . . .	30
check_optional_prob_vector . . . . .	31
check_optional_scalar_character . . . . .	32
check_optional_unit_open_vector . . . . .	33
check_pos_double_scalar . . . . .	33
check_pos_double_vector . . . . .	34
check_pos_integer_scalar . . . . .	35
check_prob_scalar . . . . .	36
check_prob_vector . . . . .	36
check_scalar_character . . . . .	37
check_scalar_logical . . . . .	38
check_tabular . . . . .	39
check_unit_open_scalar . . . . .	39
check_unit_open_vector . . . . .	40

clean_colnames . . . . .	41
clean_int . . . . .	41
clean_names . . . . .	42
clean_posint . . . . .	43
col256 . . . . .	44
collapse_head . . . . .	44
dbg . . . . .	45
ddSci . . . . .	46
double_scalar . . . . .	47
enum . . . . .	47
fmt . . . . .	48
fmt_gradient . . . . .	49
format_trace . . . . .	50
get_msg_sink . . . . .	51
get_output_type . . . . .	51
get_verbosity . . . . .	52
highlight . . . . .	53
info . . . . .	53
integer_scalar . . . . .	54
labelify . . . . .	54
logical_scalar . . . . .	55
match_arg . . . . .	56
msg . . . . .	56
msgdone . . . . .	58
msgstart . . . . .	58
nonneg_double_scalar . . . . .	59
nonneg_double_vector . . . . .	60
nonneg_integer_scalar . . . . .	60
optional . . . . .	61
optional_character_scalar . . . . .	61
optional_double_scalar . . . . .	62
optional_integer_scalar . . . . .	62
optional_logical_scalar . . . . .	63
optional_nonneg_double_scalar . . . . .	63
optional_nonneg_double_vector . . . . .	64
optional_nonneg_integer_scalar . . . . .	64
optional_pos_double_scalar . . . . .	65
optional_pos_double_vector . . . . .	65
optional_pos_integer_scalar . . . . .	66
optional_prob_scalar . . . . .	66
optional_prob_vector . . . . .	67
optional_unit_open_scalar . . . . .	67
optional_unit_open_vector . . . . .	68
plain . . . . .	68
pos_double_scalar . . . . .	69
pos_double_vector . . . . .	69
pos_integer_scalar . . . . .	70
printf . . . . .	70

printls . . . . .	71
prob_scalar . . . . .	73
prob_vector . . . . .	73
repr . . . . .	74
repr_ls . . . . .	74
rtemis_colors . . . . .	76
set_msg_sink . . . . .	77
strip_ansi . . . . .	78
success . . . . .	79
test_inherits . . . . .	79
unit_open_scalar . . . . .	80
unit_open_vector . . . . .	80
warn . . . . .	81
with_msg_sink . . . . .	82
<b>Index</b>	<b>83</b>

---

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]

Authors:

- 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	<i>Abbreviate object class name</i>
------------------	-------------------------------------

---

**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)
```

---

abort	<i>Dual-channel error signal</i>
-------	----------------------------------

---

**Description**

Signals a condition AND optionally writes a styled event line to the operator console. The two channels carry complementary information so nothing is duplicated:

**Usage**

```
abort(..., class = NULL, parent = NULL, verbosity = NULL, package = NULL)
```

## Arguments

...	Message components, concatenated with no separator.
class	Character vector: Additional condition classes (prepended to the base <code>c("rtemis_error", "error", "condition")</code> ).
parent	Condition or NULL: Wrapped parent condition. Its message is echoed to the console (when verbosity allows) and stored on the signalled condition as <code>\$parent</code> .
verbosity	Integer or NULL: Overrides <code>get_verbosity()</code> for the console echo. Set to <code>0L</code> to suppress the echo without affecting the signalled condition.
package	Character or NULL: Package name for verbosity override.

## Details

- **Console echo** (when verbosity allows): the most-specific condition class plus the caller bracket - a structured "what failed, where" line. Falls back to "rtemis\_error" when `class = NULL` so the line is always namespace-tagged and recognizable as ours.
- **Condition \$message**: the corrective human-readable text passed via ... Plain text with all ANSI escapes stripped, so it is safe to serialize into JSON, HTML, or any other ANSI-unaware sink (e.g. browser-side error display), and is what `conditionMessage()` / R's default error printer / `tryCatch(error = ...)` handlers see.

Use `class` to add wire-protocol-specific condition classes that callers can catch via `tryCatch()`. The base classes "rtemis\_error", "error", and "condition" are always added.

The condition also carries `$trace` - a pairlist of `sys.calls()` captured at the abort site, with `abort()`'s own frame trimmed. Unlike base R's `traceback()` (which only sees `.Traceback`, populated only when an error reaches the top-level uncaught), `$trace` survives `tryCatch()` and travels with the condition - so server-side handlers can ship the stack to a browser-side debug pane, or callers can call `format_trace()` to print it.

## Value

Does not return - always signals a condition via `stop()`.

## Author(s)

EDG

## Examples

```
## Not run:
abort("Could not parse ", "hyperparameters", ".",
      class = "rtemislive_invalid_params")

## End(Not run)
```

---

ansi256_to_hex	<i>Convert ANSI 256 color code to HEX</i>
----------------	---

---

**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)
```

---

<b>bold</b>	<i>Make text bold</i>
-------------	-----------------------

---

**Description**

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

**Usage**

```
bold(text, reset_code = "\033[22m", output_type = c("ansi", "html", "plain"))
```

**Arguments**

text	Character: Text to make bold.
reset_code	Character: ANSI reset code to use after formatting.
output_type	Character: Output type ("ansi", "html", "plain").

**Value**

Character: Formatted text with bold styling

**Author(s)**

EDG

**Examples**

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

---

 bounded\_double\_property

*Create a bounded double S7 property*


---

**Description**

Returns a new\_property() for a double scalar constrained to a given interval. Useful for bounds not covered by the pre-built properties.

**Usage**

```
bounded_double_property(
  lower = -Inf,
  upper = Inf,
  lower_open = FALSE,
  upper_open = FALSE,
  nullable = FALSE
)
```

**Arguments**

lower	Numeric scalar. Lower bound. Default -Inf.
upper	Numeric scalar. Upper bound. Default Inf.
lower_open	Logical scalar. If TRUE, lower bound is exclusive (lower, ...]. Default FALSE.
upper_open	Logical scalar. If TRUE, upper bound is exclusive [..., upper). Default FALSE.
nullable	Logical scalar. If TRUE, NULL is also accepted. Default FALSE.

**Value**

An S7 property object.

**Author(s)**

EDG

**Examples**

```
# Learning rate in (0, 1]
lr_prop <- bounded_double_property(0, 1, lower_open = TRUE)
```

---

character_scalar	<i>Non-empty character scalar S7 property</i>
------------------	---

---

**Description**

S7 property accepting a single non-NA, non-empty (after trimming whitespace) string.

**Usage**

```
character_scalar
```

**Value**

An S7 property object.

**Author(s)**

EDG

---

check_character	<i>Check character</i>
-----------------	------------------------

---

**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\_character\_scalar

*Check character scalar*

---

**Description**

Check character scalar

**Usage**

```
check_character_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Character: Value to check. Must be a single non-NA, non-empty string.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_character_scalar("hello")
# Throw error:
try(check_character_scalar(""))
try(check_character_scalar(NA_character_))
try(check_character_scalar(c("a", "b")))
```

---

check_data.table	<i>Check data.table</i>
------------------	-------------------------

---

**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	<b>rtemis.core</b> <i>internal: Dependencies check</i>
--------------------	--

---

**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\_double\_scalar    *Check double scalar*

---

**Description**

Check double scalar

**Usage**

```
check_double_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a single non-NA number (integer inputs are accepted).
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_double_scalar(3.14)
check_double_scalar(1L)
# Throw error:
try(check_double_scalar(NA_real_))
try(check_double_scalar(c(1.0, 2.0)))
```

---

check_enum	<i>Check if value is in set of allowed values</i>
------------	---

---

**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_file_exists	<i>Check file exists</i>
-------------------	--------------------------

---

**Description**

Check file exists

**Usage**

```
check_file_exists(file)
```

**Arguments**

file	Character: Path to file to check.
------	-----------------------------------

**Value**

Throws an error if checks fail. Returns the normalized file path invisibly if checks pass.

**Author(s)**

EDG

**Examples**

```
## Not run:
check_file_exists("path/to/file.txt")

## End(Not run)
```

---

check_float01exc	<i>Check float between 0 and 1, exclusive</i>
------------------	---

---

**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	<i>Check float between 0 and 1, inclusive</i>
------------------	---

---

**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	<i>Check float greater than or equal to 0</i>
-----------------	---

---

**Description**

Checks if an input is a numeric vector containing non-negative ( $\geq 0$ ) 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	<i>Check positive float</i>
----------------	-----------------------------

---

**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	<i>Check float in (0, 1]</i>
-----------------	------------------------------

---

**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  $-1 \leq x \leq 1$

**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\_integer\_scalar *Check integer scalar*

---

**Description**

Check integer scalar

**Usage**

```
check_integer_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a single non-NA whole number.
arg_name	Character: Argument name to use in error messages.

**Details**

Accepts any single numeric value that is a whole number. Integer-typed inputs (1L) and double-typed whole numbers (1, 100) are both accepted for user convenience.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_integer_scalar(5L)
check_integer_scalar(100)
# Throw error:
try(check_integer_scalar(1.5))
try(check_integer_scalar(NA_integer_))
```

---

`check_logical`*Check logical*

---

**Description**

Check logical

**Usage**

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

**Arguments**

<code>x</code>	Vector to check.
<code>allow_null</code>	Logical: If TRUE, NULL values are allowed and return early.
<code>arg_name</code>	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\_logical\_scalar *Check logical scalar*

---

**Description**

Check logical scalar

**Usage**

```
check_logical_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x                    Logical: Value to check. Must be a single non-NA TRUE or FALSE.  
arg\_name            Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_logical_scalar(TRUE)
check_logical_scalar(FALSE)
# Throw error:
try(check_logical_scalar(NA))
try(check_logical_scalar(1L))
try(check_logical_scalar(c(TRUE, FALSE)))
```

---

check\_nonneg\_double\_scalar  
*Check non-negative double scalar*

---

**Description**

Check non-negative double scalar

**Usage**

```
check_nonneg_double_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a single finite number greater than or equal to zero.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_nonneg_double_scalar(0)
check_nonneg_double_scalar(5)
# Throw error:
try(check_nonneg_double_scalar(-0.001))
try(check_nonneg_double_scalar(Inf))
```

---

check\_nonneg\_double\_vector

*Check non-negative double vector*

---

**Description**

Check non-negative double vector

**Usage**

```
check_nonneg_double_vector(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a non-empty vector with all elements finite, greater than or equal to zero, and no NAs.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

## Examples

```
check_nonneg_double_vector(c(0, 1, 2.5))
# Throw error:
try(check_nonneg_double_vector(c(-1, 0, 1)))
try(check_nonneg_double_vector(c(1, Inf)))
```

---

check_numeric	<i>Check numeric</i>
---------------	----------------------

---

## Description

Checks that `x` is numeric. Uses `is.numeric()`, which accepts both "double" and "integer" inputs - the right semantics for "is this a number?". Prefer this over `check_inherits(x, "numeric")`, which rejects integers because their literal class is "integer", not "numeric" (a long-standing R/S3 quirk). This trips up wire-format callers: `jsonlite::fromJSON("1")` returns an integer, and the value would then fail `inherits(., "numeric")` despite being a perfectly good number.

## Usage

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

## Arguments

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

## Value

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

## Author(s)

EDG

## Examples

```
check_numeric(1L)
check_numeric(1.5)
check_numeric(c(1, 2, 3))
# Throws error:
try(check_numeric("1"))
try(check_numeric(TRUE))
```

---

check\_optional\_character\_scalar  
*Check optional character scalar*

---

**Description**

Check optional character scalar

**Usage**

```
check_optional_character_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Character: Value to check. Must be NULL or a single non-NA, non-empty string.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_character_scalar(NULL)
check_optional_character_scalar("hello")
# Throw error:
try(check_optional_character_scalar(""))
try(check_optional_character_scalar(c("a", "b")))
```

---

check\_optional\_double\_scalar  
*Check optional double scalar*

---

**Description**

Check optional double scalar

**Usage**

```
check_optional_double_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a single non-NA number.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_double_scalar(NULL)
check_optional_double_scalar(2.5)
# Throw error:
try(check_optional_double_scalar(NA_real_))
```

---

check\_optional\_integer\_scalar  
*Check optional integer scalar*

---

**Description**

Check optional integer scalar

**Usage**

```
check_optional_integer_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a single non-NA whole number.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_integer_scalar(NULL)
check_optional_integer_scalar(10L)
# Throw error:
try(check_optional_integer_scalar(1.5))
```

---

```
check_optional_logical_scalar
  Check optional logical scalar
```

---

**Description**

Check optional logical scalar

**Usage**

```
check_optional_logical_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Logical: Value to check. Must be NULL or a single non-NA TRUE or FALSE.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_logical_scalar(NULL)
check_optional_logical_scalar(FALSE)
# Throw error:
try(check_optional_logical_scalar(NA))
```

---

check\_optional\_nonneg\_double\_scalar  
*Check optional non-negative double scalar*

---

**Description**

Check optional non-negative double scalar

**Usage**

```
check_optional_nonneg_double_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a single finite number greater than or equal to zero.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_nonneg_double_scalar(NULL)
check_optional_nonneg_double_scalar(0)
# Throw error:
try(check_optional_nonneg_double_scalar(-1))
```

---

check\_optional\_nonneg\_double\_vector  
*Check optional non-negative double vector*

---

**Description**

Check optional non-negative double vector

**Usage**

```
check_optional_nonneg_double_vector(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a non-empty vector with all elements finite, greater than or equal to zero, and no NAs.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_nonneg_double_vector(NULL)
check_optional_nonneg_double_vector(c(0, 1, 5))
# Throw error:
try(check_optional_nonneg_double_vector(c(-1, 0)))
```

---

check\_optional\_pos\_double\_scalar

*Check optional positive double scalar*

---

**Description**

Check optional positive double scalar

**Usage**

```
check_optional_pos_double_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a single finite number strictly greater than zero.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_pos_double_scalar(NULL)
check_optional_pos_double_scalar(2.5)
# Throw error:
try(check_optional_pos_double_scalar(0))
```

---

check\_optional\_pos\_double\_vector  
*Check optional positive double vector*

---

**Description**

Check optional positive double vector

**Usage**

```
check_optional_pos_double_vector(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a non-empty vector with all elements finite, strictly greater than zero, and no NAs.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_pos_double_vector(NULL)
check_optional_pos_double_vector(c(0.5, 2))
# Throw error:
try(check_optional_pos_double_vector(c(0, 1)))
```

check\_optional\_pos\_integer\_scalar

*Check optional positive integer scalar*

---

**Description**

Check optional positive integer scalar

**Usage**

```
check_optional_pos_integer_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a single non-NA whole number greater than zero.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_pos_integer_scalar(NULL)
check_optional_pos_integer_scalar(5L)
# Throw error:
try(check_optional_pos_integer_scalar(0))
```

---

check\_optional\_prob\_scalar

*Check optional probability scalar*

---

**Description**

Check optional probability scalar

**Usage**

```
check_optional_prob_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a single finite number in [0, 1].
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_prob_scalar(NULL)
check_optional_prob_scalar(0.5)
# Throw error:
try(check_optional_prob_scalar(2.0))
```

---

check\_optional\_prob\_vector  
*Check optional probability vector*

---

**Description**

Check optional probability vector

**Usage**

```
check_optional_prob_vector(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a non-empty vector with all elements in [0, 1] and no NAs.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_prob_vector(NULL)
check_optional_prob_vector(c(0.2, 0.8))
# Throw error:
try(check_optional_prob_vector(c(0.5, 2)))
```

---

check\_optional\_scalar\_character  
*Check Optional Scalar Character*

---

**Description**

Check Optional Scalar Character

**Usage**

```
check_optional_scalar_character(x, arg_name = deparse(substitute(x)))
```

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

---

check\_optional\_unit\_open\_vector  
*Check optional open-unit-interval vector*

---

**Description**

Check optional open-unit-interval vector

**Usage**

```
check_optional_unit_open_vector(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Optional Numeric: Value to check. Must be NULL or a non-empty vector with all elements strictly in (0, 1) and no NAs.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_optional_unit_open_vector(NULL)
check_optional_unit_open_vector(c(0.1, 0.9))
# Throw error:
try(check_optional_unit_open_vector(c(0, 0.5)))
```

---

check\_pos\_double\_scalar  
*Check positive double scalar*

---

**Description**

Check positive double scalar

**Usage**

```
check_pos_double_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a single finite number strictly greater than zero.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_pos_double_scalar(0.001)
check_pos_double_scalar(100)
# Throw error:
try(check_pos_double_scalar(0))
try(check_pos_double_scalar(-1))
try(check_pos_double_scalar(Inf))
```

---

check\_pos\_double\_vector

*Check positive double vector*

---

**Description**

Check positive double vector

**Usage**

```
check_pos_double_vector(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a non-empty vector with all elements finite, strictly greater than zero, and no NAs.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_pos_double_vector(c(0.1, 1, 10))
# Throw error:
try(check_pos_double_vector(c(0, 1)))
try(check_pos_double_vector(c(1, Inf)))
```

---

check\_pos\_integer\_scalar

*Check positive integer scalar*

---

**Description**

Check positive integer scalar

**Usage**

```
check_pos_integer_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a single non-NA whole number greater than zero.
arg_name	Character: Argument name to use in error messages.

**Details**

Accepts any single numeric value that is a whole number strictly greater than zero. Integer-typed inputs (1L) and double-typed whole numbers (1, 100) are both accepted for user convenience.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_pos_integer_scalar(1L)
check_pos_integer_scalar(10)
# Throw error:
try(check_pos_integer_scalar(0))
try(check_pos_integer_scalar(-1L))
try(check_pos_integer_scalar(1.5))
```

---

check\_prob\_scalar      *Check probability scalar*

---

**Description**

Check probability scalar

**Usage**

```
check_prob_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x                      Numeric: Value to check. Must be a single finite number in  $[0, 1]$ .  
arg\_name              Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_prob_scalar(0)
check_prob_scalar(0.5)
check_prob_scalar(1)
# Throw error:
try(check_prob_scalar(1.5))
try(check_prob_scalar(-0.1))
```

---

check\_prob\_vector      *Check probability vector*

---

**Description**

Check probability vector

**Usage**

```
check_prob_vector(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a non-empty vector with all elements in $[0, 1]$ and no NAs.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_prob_vector(c(0, 0.5, 1))
# Throw error:
try(check_prob_vector(c(0.5, 1.5)))
try(check_prob_vector(c(0.5, NA)))
```

---

check\_scalar\_character

*Check Scalar Character*

---

**Description**

Check Scalar Character

**Usage**

```
check_scalar_character(x, arg_name = deparse(substitute(x)))
```

**Arguments**

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

**Value**

Called for side effects.

**Author(s)**

EDG

**Examples**

```
check_scalar_character("hello", "my_arg")
# Throw error:
try(check_scalar_character(c("hello", "world"), "my_arg"))
try(check_scalar_character(123, "my_arg"))
```

---

check\_scalar\_logical *Check Scalar Logical*

---

**Description**

Check Scalar Logical

**Usage**

```
check_scalar_logical(x, arg_name = deparse(substitute(x)))
```

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

---

check_tabular	<i>Check object is tabular</i>
---------------	--------------------------------

---

**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)))
```

---

check_unit_open_scalar	<i>Check open-unit-interval scalar</i>
------------------------	--

---

**Description**

Check open-unit-interval scalar

**Usage**

```
check_unit_open_scalar(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x                    Numeric: Value to check. Must be a single finite number strictly in (0, 1).  
arg\_name            Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_unit_open_scalar(0.5)
# Throw error:
try(check_unit_open_scalar(0))
try(check_unit_open_scalar(1))
```

---

check\_unit\_open\_vector

*Check open-unit-interval vector*

---

**Description**

Check open-unit-interval vector

**Usage**

```
check_unit_open_vector(x, arg_name = deparse(substitute(x)))
```

**Arguments**

x	Numeric: Value to check. Must be a non-empty vector with all elements strictly in (0, 1) and no NAs.
arg_name	Character: Argument name to use in error messages.

**Value**

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

**Author(s)**

EDG

**Examples**

```
check_unit_open_vector(c(0.2, 0.5, 0.9))
# Throw error:
try(check_unit_open_vector(c(0, 0.5)))
try(check_unit_open_vector(c(0.5, 1)))
```

---

clean_colnames	<i>Clean column names</i>
----------------	---------------------------

---

**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	<i>Clean integer input</i>
-----------	----------------------------

---

**Description**

Clean integer input

**Usage**

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

**Arguments**

x                      Double or integer vector to check.  
arg\_name              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	<i>Check positive integer</i>
--------------	-------------------------------

---

**Description**

Check positive integer

**Usage**

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

**Arguments**

x	Integer vector.
allow_na	Logical: If TRUE, NAs are excluded before checking. If FALSE (default), NAs trigger an error.
arg_name	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")
```

---

collapse\_head *Collapse head of vector with commas followed by ellipsis*

---

**Description**

Collapse head of vector with commas followed by ellipsis

**Usage**

```
collapse_head(x, maxlength = 6L, format_fn = identity)
```

**Arguments**

x	Vector: Input whose first elements are shown.
maxlength	Integer: Maximum number of elements to show before truncating with an ellipsis. Use -1 to show all.
format_fn	Function: Formatting function applied to each element.

**Details**

Used, for example, by `repr_ls`

**Value**

Character.

**Author(s)**

EDG

**Examples**

```
collapse_head(98054:99890, maxlength = 5L)
collapse_head(
  c("mango", "banana", "tangerine", "sugar", "ackee", "cocoa bean"),
  maxlength = 3L, format_fn = toupper
)
```

---

dbg

*Debug log message*

---

**Description**

Muted log message gated at verbosity  $\geq$  2L. Use for development / troubleshooting output that should not appear in normal operation.

**Usage**

```
dbg(..., verbosity = NULL, package = NULL)
```

**Arguments**

<code>...</code>	Message components, concatenated with no separator.
<code>verbosity</code>	Integer or NULL: Overrides <code>get_verbosity()</code> when supplied.
<code>package</code>	Character or NULL: Package name for verbosity override.

**Details**

Named `dbg()` rather than `debug()` to avoid shadowing `base::debug()` - the R debugger entry point.

**Value**

Invisible NULL.

**Author(s)**

EDG

**Examples**

```
dbg("payload bytes: ", 1234L)
```

---

`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**

<code>x</code>	Vector of numbers
<code>decimal_places</code>	Integer: Return this many decimal places.
<code>hi</code>	Float: Threshold at or above which scientific notation is used.
<code>as_numeric</code>	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 <code>as_numeric = FALSE</code> , 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"
```

---

double_scalar	<i>Double scalar S7 property</i>
---------------	----------------------------------

---

**Description**

S7 property accepting a single non-NA double value.

**Usage**

```
double_scalar
```

**Value**

An S7 property object.

**Author(s)**

EDG

---

enum	<i>Create an enum S7 property</i>
------	-----------------------------------

---

**Description**

Returns a `new_property()` for a character scalar constrained to a fixed set of allowed values.

**Usage**

```
enum(values, default = NULL, nullable = FALSE)
```

**Arguments**

values	Character: Allowed values.
default	Optional Character: Default value.
nullable	Logical scalar. If TRUE, NULL is also accepted. Default FALSE.

**Value**

An S7 property object.

**Author(s)**

EDG

**Examples**

```
type_prop <- enum(c("string", "number", "boolean"), default = "string")
```

---

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,  
  reset_code = "\033[0m",  
  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.
reset_code	Character: ANSI reset code to use after formatting.
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()
```

---

format\_trace

*Pretty-print a captured call trace*

---

**Description**

Formats the `$trace` carried by an `rtemis_error` condition (see [abort\(\)](#)) as a numbered, one-line-per-frame string. Most-recent frame at the bottom, matching base R's [traceback\(\)](#) convention. Each frame is deparsed with a single-line cap so long calls stay readable; no styling is applied, so the output is safe for any sink (terminal, JSON, HTML).

**Usage**

```
format_trace(trace, max_width = 80L)
```

**Arguments**

<code>trace</code>	pairlist of calls, as captured by <a href="#">abort()</a> on <code>cond\$trace</code> . Passing the condition itself also works - the trace is extracted via <code>cond\$trace</code> .
<code>max_width</code>	Integer: Max characters per deparsed line. Longer calls are truncated with a trailing ellipsis.

**Value**

Character scalar with one frame per `\n`-separated line, newest frame last. `""` if the trace is empty or `NULL`.

**Author(s)**

EDG

**Examples**

```
## Not run:
cond <- tryCatch(
  check_numeric("oops"),
  error = identity
)
cat(format_trace(cond), "\n")

## End(Not run)
```

---

get_msg_sink	<i>Get the current rtemis message sink</i>
--------------	--

---

**Description**

Get the current rtemis message sink

**Usage**

```
get_msg_sink()
```

**Value**

The currently registered sink function, or NULL if none is set.

**Author(s)**

EDG

**See Also**

[set\\_msg\\_sink\(\)](#), [with\\_msg\\_sink\(\)](#).

---

get_output_type	<i>Get output type</i>
-----------------	------------------------

---

**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()
```

---

get_verbosity	<i>Resolve the current logging verbosity</i>
---------------	--

---

**Description**

Reads `getOption("<package>.verbosity")` first when package is supplied, falling back to `getOption("rtemis.verbosity")` and finally to 1L. Levels: 0L silent, 1L info/warn/success/abort console echo, 2L includes debug.

**Usage**

```
get_verbosity(package = NULL)
```

**Arguments**

package        Character or NULL: Optional package-specific override.

**Value**

Integer scalar verbosity level.

**Author(s)**

EDG

**Examples**

```
get_verbosity()
```

---

highlight	<i>Highlight text</i>
-----------	-----------------------

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

Character: Formatted text with highlight.

**Author(s)**

EDG

**Examples**

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

---

info	<i>Informational log message</i>
------	----------------------------------

---

**Description**

Styled informational message, routed through `msg()` so it carries the shared datetime + caller prefix. Fires when verbosity is at least 1L.

**Usage**

```
info(..., verbosity = NULL, package = NULL)
```

**Arguments**

<code>...</code>	Message components, concatenated with no separator.
<code>verbosity</code>	Integer or NULL: Overrides <code>get_verbosity()</code> when supplied.
<code>package</code>	Character or NULL: Package name for verbosity override lookup (e.g. "rtemis.server").

**Value**

Invisible NULL.

**Author(s)**

EDG

**Examples**

```
info("Server started on port ", 8080L)
```

---

integer_scalar	<i>Integer scalar S7 property</i>
----------------	-----------------------------------

---

**Description**

S7 property accepting a single non-NA integer value (must be integer type, e.g. 1L).

**Usage**

```
integer_scalar
```

**Value**

An S7 property object.

**Author(s)**

EDG

---

labelify	<i>Format text for label printing</i>
----------	---------------------------------------

---

**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)
```

---

logical_scalar	<i>Logical scalar S7 property</i>
----------------	-----------------------------------

---

**Description**

S7 property accepting a single non-NA logical value.

**Usage**

```
logical_scalar
```

**Value**

An S7 property object.

**Author(s)**

EDG

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.")
```

---

*nonneg\_double\_scalar*    *Non-negative double scalar S7 property*

---

**Description**

S7 property accepting a single finite double greater than or equal to zero, i.e. in  $[0, \infty)$ .

**Usage**

```
nonneg_double_scalar
```

**Value**

An S7 property object.

**Author(s)**

EDG

---

nonneg\_double\_vector *Non-negative double vector S7 property*

---

**Description**

S7 property accepting a non-empty double vector with all elements finite, greater than or equal to zero, and no NAs.

**Usage**

nonneg\_double\_vector

**Value**

An S7 property object.

**Author(s)**

EDG

---

nonneg\_integer\_scalar *Non-negative integer scalar S7 property*

---

**Description**

S7 property accepting a single non-NA integer value greater than or equal to zero, i.e. in  $[0, \infty)$  (e.g. 0L, 1L).

**Usage**

nonneg\_integer\_scalar

**Value**

An S7 property object.

**Author(s)**

EDG

---

`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.

**Details**

This should be used when the S7 class already includes all the necessary validation for the non-NULL case. Otherwise, create a new S7 property with appropriate validator using `S7::new_property()`.

**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)
```

---

`optional_character_scalar`*Optional non-empty character scalar S7 property*

---

**Description**

S7 property accepting NULL or a single non-NA, non-empty (after trimming whitespace) string.

**Usage**

```
optional_character_scalar
```

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_double\_scalar

*Optional double scalar S7 property*

---

**Description**

S7 property accepting NULL or a single non-NA double value.

**Usage**

optional\_double\_scalar

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_integer\_scalar

*Optional integer scalar S7 property*

---

**Description**

S7 property accepting NULL or a single non-NA integer value (must be integer type, e.g. 1L).

**Usage**

optional\_integer\_scalar

**Value**

An S7 property object.

**Author(s)**

EDG

---

`optional_logical_scalar`

*Optional logical scalar S7 property*

---

**Description**

S7 property accepting NULL or a single non-NA logical value.

**Usage**

`optional_logical_scalar`

**Value**

An S7 property object.

**Author(s)**

EDG

---

`optional_nonneg_double_scalar`

*Optional non-negative double scalar S7 property*

---

**Description**

S7 property accepting NULL or a single finite double greater than or equal to zero.

**Usage**

`optional_nonneg_double_scalar`

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_nonneg\_double\_vector

*Optional non-negative double vector S7 property*

---

**Description**

S7 property accepting NULL or a non-empty double vector with all elements finite, greater than or equal to zero, and no NAs.

**Usage**

optional\_nonneg\_double\_vector

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_nonneg\_integer\_scalar

*Optional non-negative integer scalar S7 property*

---

**Description**

S7 property accepting NULL or a single non-NA integer value greater than or equal to zero.

**Usage**

optional\_nonneg\_integer\_scalar

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_pos\_double\_scalar

*Optional positive double scalar S7 property*

---

**Description**

S7 property accepting NULL or a single finite double strictly greater than zero.

**Usage**

optional\_pos\_double\_scalar

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_pos\_double\_vector

*Optional positive double vector S7 property*

---

**Description**

S7 property accepting NULL or a non-empty double vector with all elements finite, strictly greater than zero, and no NAs.

**Usage**

optional\_pos\_double\_vector

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_pos\_integer\_scalar

*Optional positive integer scalar S7 property*

---

**Description**

S7 property accepting NULL or a single non-NA integer value strictly greater than zero.

**Usage**

optional\_pos\_integer\_scalar

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_prob\_scalar

*Optional probability scalar S7 property*

---

**Description**

S7 property accepting NULL or a single finite double in  $[0, 1]$ .

**Usage**

optional\_prob\_scalar

**Value**

An S7 property object.

**Author(s)**

EDG

---

`optional_prob_vector`    *Optional probability vector S7 property*

---

**Description**

S7 property accepting NULL or a non-empty double vector with all elements in  $[0, 1]$  and no NAs.

**Usage**

`optional_prob_vector`

**Value**

An S7 property object.

**Author(s)**

EDG

---

`optional_unit_open_scalar`  
*Optional open-unit-interval scalar S7 property*

---

**Description**

S7 property accepting NULL or a single finite double strictly in  $(0, 1)$ .

**Usage**

`optional_unit_open_scalar`

**Value**

An S7 property object.

**Author(s)**

EDG

---

optional\_unit\_open\_vector

*Optional open-unit-interval vector S7 property*

---

**Description**

S7 property accepting NULL or a non-empty double vector with all elements strictly in  $(0, 1)$  and no NAs.

**Usage**

optional\_unit\_open\_vector

**Value**

An S7 property object.

**Author(s)**

EDG

---

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"))
```

---

pos_double_scalar	<i>Positive double scalar S7 property</i>
-------------------	---

---

**Description**

S7 property accepting a single finite double strictly greater than zero, i.e. in  $(0, \infty)$ .

**Usage**

pos\_double\_scalar

**Value**

An S7 property object.

**Author(s)**

EDG

---

pos_double_vector	<i>Positive double vector S7 property</i>
-------------------	---

---

**Description**

S7 property accepting a non-empty double vector with all elements finite, strictly greater than zero, and no NAs.

**Usage**

pos\_double\_vector

**Value**

An S7 property object.

**Author(s)**

EDG

---

pos_integer_scalar	<i>Positive integer scalar S7 property</i>
--------------------	--

---

**Description**

S7 property accepting a single non-NA integer value strictly greater than zero (e.g. 1L).

**Usage**

```
pos_integer_scalar
```

**Value**

An S7 property object.

**Author(s)**

EDG

---

printf	<i>Print data frame</i>
--------	-------------------------

---

**Description**

Pretty print a data frame

**Usage**

```
printf(  
  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 <code>ddSci</code> . 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 <code>collapse_head()</code> 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")
```

---

`prob_scalar`*Probability scalar S7 property*

---

**Description**

S7 property accepting a single finite double in  $[0, 1]$ .

**Usage**`prob_scalar`**Value**

An S7 property object.

**Author(s)**

EDG

---

`prob_vector`*Probability vector S7 property*

---

**Description**

S7 property accepting a non-empty double vector with all elements in  $[0, 1]$  and no NAs.

**Usage**`prob_vector`**Value**

An S7 property object.

**Author(s)**

EDG

---

repr	<i>String representation</i>
------	------------------------------

---

**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	<i>Show list as formatted string</i>
---------	--------------------------------------

---

**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 collapse_head() 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
```

**Value**

Named character vector of hex color codes.

**Author(s)**

EDG

**Examples**

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

---

set_msg_sink	<i>Set the rtemis message sink</i>
--------------	------------------------------------

---

### Description

When `set`, `msg()`, `msg0()`, `msgstart()`, and `msgdone()` forward their structured output through `sink` instead of writing to the R console. Used by `rtemis.server` to capture training-time messages and forward them over a WebSocket connection. Pass `NULL` to restore default console output.

### Usage

```
set_msg_sink(sink)
```

### Arguments

`sink`            Function or `NULL`.

### Details

The `sink` function is called once per message with a single argument: a list with fields

- `text`: character. The formatted message body (no datetime prefix).
- `caller`: character or `NA`. Calling function as identified by `format_caller()`.
- `ts`: character. Formatted timestamp ("%Y-%m-%d %H:%M:%S").
- `level`: character. One of "info" (`msg/msg0`), "start" (`msgstart`), or "done" (`msgdone`).

Producers may include **additional** fields in the list, which sinks should ignore when not understood. In particular, `rtemis`'s training observability (see its `specs/observability.md`) emits execution-graph node events through the sink with these extra fields:

- `node_id`: character. Unique id of the execution-graph node.
- `parent_id`: character or `NA`. Parent node id (for nesting).
- `kind`: character. Node kind (e.g. "tune", "grid\_cell", "train\_alg").
- `status`: character. "start", "done", "error", or "aborted".
- `current`, `total`: integer or `NULL`. Progress counters.

These are additive; sinks that only read the base fields keep working.

When a sink is set, the console output path is **skipped** for affected calls. Errors thrown by the sink propagate to the caller of `msg()`.

### Value

Previous sink (function or `NULL`), invisibly.

### Author(s)

EDG

**See Also**

`get_msg_sink()`, `with_msg_sink()`.

**Examples**

```
captured <- list()
set_msg_sink(function(m) captured[[length(captured) + 1L]] <<- m)
# msg("hello world")      # would append to `captured`
set_msg_sink(NULL)        # restore console output
```

---

strip\_ansi

*Strip ANSI escape sequences from a string*

---

**Description**

Removes the SGR / CSI escapes commonly produced by `fmt()` (and by any other tool that writes coloured terminal output). Safe to call on plain input - returns it unchanged.

**Usage**

```
strip_ansi(x)
```

**Arguments**

x                   Character: Input.

**Value**

Character: x with ANSI escapes removed.

**Author(s)**

EDG

**Examples**

```
strip_ansi(fmt("hi", col = "red"))
```

---

success	<i>Success log message</i>
---------	----------------------------

---

**Description**

Styled success message. Fires when verbosity is at least 1L.

**Usage**

```
success(..., verbosity = NULL, package = NULL)
```

**Arguments**

...	Message components, concatenated with no separator.
verbosity	Integer or NULL: Overrides <code>get_verbosity()</code> when supplied.
package	Character or NULL: Package name for verbosity override.

**Value**

Invisible NULL.

**Author(s)**

EDG

**Examples**

```
success("Job ", "abc123", " complete")
```

---

test_inherits	<i>Check class of object</i>
---------------	------------------------------

---

**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)
```

---

unit_open_scalar	<i>Open-unit-interval scalar S7 property</i>
------------------	--

---

**Description**

S7 property accepting a single finite double strictly in  $(0, 1)$ .

**Usage**

```
unit_open_scalar
```

**Value**

An S7 property object.

**Author(s)**

EDG

---

unit_open_vector	<i>Open-unit-interval vector S7 property</i>
------------------	--

---

**Description**

S7 property accepting a non-empty double vector with all elements strictly in  $(0, 1)$  and no NAs.

**Usage**

```
unit_open_vector
```

**Value**

An S7 property object.

**Author(s)**

EDG

---

warn	<i>Warning log message</i>
------	----------------------------

---

**Description**

Styled non-fatal message. By default emits a soft styled message via `msg()`; with `use_warning = TRUE`, calls `warning()` so callers can catch via `tryCatch(..., warning = handler)`.

**Usage**

```
warn(..., use_warning = FALSE, verbosity = NULL, package = NULL)
```

**Arguments**

<code>...</code>	Message components, concatenated with no separator.
<code>use_warning</code>	Logical: If TRUE, signal an R warning condition instead of (or in addition to) writing a styled message.
<code>verbosity</code>	Integer or NULL: Overrides <code>get_verbosity()</code> when supplied.
<code>package</code>	Character or NULL: Package name for verbosity override.

**Value**

Invisible NULL.

**Author(s)**

EDG

**Examples**

```
warn("Disk usage at ", 92L, "%")
```

---

with_msg_sink	<i>Run code with a temporary message sink</i>
---------------	---

---

**Description**

Sets sink for the duration of code, restoring the previous sink on exit (including on error). Useful in tests and for short-lived capture.

**Usage**

```
with_msg_sink(sink, code)
```

**Arguments**

sink	Sink function or NULL.
code	Code to run.

**Value**

The value returned by code.

**Author(s)**

EDG

**See Also**

[set\\_msg\\_sink\(\)](#), [get\\_msg\\_sink\(\)](#).

**Examples**

```
captured <- list()
with_msg_sink(
  function(m) captured[[length(captured) + 1L]] <<- m,
  {
    # any msg() / msg0() / msgstart() / msgdone() calls in here are captured
  }
)
```

# Index

abbreviate\_class, 5  
abort, 5  
abort(), 50  
ansi256\_to\_hex, 7  
  
base::debug(), 45  
bold, 7  
bounded\_double\_property, 8  
  
character\_scalar, 9  
check\_character, 9  
check\_character\_scalar, 10  
check\_data.table, 11  
check\_dependencies, 11  
check\_double\_scalar, 12  
check\_enum, 13  
check\_file\_exists, 13  
check\_float01exc, 14  
check\_float01inc, 15  
check\_float0pos, 15  
check\_float\_neg1\_1, 18  
check\_floatpos, 16  
check\_floatpos1, 17  
check\_inherits, 18  
check\_integer\_scalar, 19  
check\_logical, 20  
check\_logical\_scalar, 21  
check\_nonneg\_double\_scalar, 21  
check\_nonneg\_double\_vector, 22  
check\_numeric, 23  
check\_optional\_character\_scalar, 24  
check\_optional\_double\_scalar, 24  
check\_optional\_integer\_scalar, 25  
check\_optional\_logical\_scalar, 26  
check\_optional\_nonneg\_double\_scalar, 27  
check\_optional\_nonneg\_double\_vector, 27  
check\_optional\_pos\_double\_scalar, 28  
check\_optional\_pos\_double\_vector, 29  
check\_optional\_pos\_integer\_scalar, 30  
check\_optional\_prob\_scalar, 30  
check\_optional\_prob\_vector, 31  
check\_optional\_scalar\_character, 32  
check\_optional\_unit\_open\_vector, 33  
check\_pos\_double\_scalar, 33  
check\_pos\_double\_vector, 34  
check\_pos\_integer\_scalar, 35  
check\_prob\_scalar, 36  
check\_prob\_vector, 36  
check\_scalar\_character, 37  
check\_scalar\_logical, 38  
check\_tabular, 39  
check\_unit\_open\_scalar, 39  
check\_unit\_open\_vector, 40  
clean\_colnames, 41  
clean\_int, 41  
clean\_names, 42  
clean\_posint, 43  
col256, 44  
collapse\_head, 44  
  
dbg, 45  
ddSci, 46, 71  
double\_scalar, 47  
  
enum, 47  
  
fmt, 48  
fmt\_gradient, 49  
format\_trace, 50  
format\_trace(), 6  
  
get\_msg\_sink, 51  
get\_msg\_sink(), 78, 82  
get\_output\_type, 51  
get\_verbosity, 52  
  
highlight, 53  
info, 53

- integer\_scalar, 54
- labelify, 54
- logical\_scalar, 55
  
- match\_arg, 56
- msg, 56
- msg0 (msg), 56
- msgdone, 58
- msgstart, 58
  
- nonneg\_double\_scalar, 59
- nonneg\_double\_vector, 60
- nonneg\_integer\_scalar, 60
  
- optional, 61
- optional\_character\_scalar, 61
- optional\_double\_scalar, 62
- optional\_integer\_scalar, 62
- optional\_logical\_scalar, 63
- optional\_nonneg\_double\_scalar, 63
- optional\_nonneg\_double\_vector, 64
- optional\_nonneg\_integer\_scalar, 64
- optional\_pos\_double\_scalar, 65
- optional\_pos\_double\_vector, 65
- optional\_pos\_integer\_scalar, 66
- optional\_prob\_scalar, 66
- optional\_prob\_vector, 67
- optional\_unit\_open\_scalar, 67
- optional\_unit\_open\_vector, 68
  
- plain, 68
- pos\_double\_scalar, 69
- pos\_double\_vector, 69
- pos\_integer\_scalar, 70
- printf, 70
- println, 71
- prob\_scalar, 73
- prob\_vector, 73
  
- repr, 74
- repr\_ls, 74
- rtemis.core (rtemis.core-package), 4
- rtemis.core-package, 4
- rtemis\_colors, 76
  
- set\_msg\_sink, 77
- set\_msg\_sink(), 51, 82
- strip\_ansi, 78
- success, 79
  
- test\_inherits, 79
- traceback(), 50
  
- unit\_open\_scalar, 80
- unit\_open\_vector, 80
  
- warn, 81
- with\_msg\_sink, 82
- with\_msg\_sink(), 51, 78