# Package 'rtemis.core'

November 2, 2026

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

**Version** 0.4.7

**Date** 2026-10-09

**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>).

**License** BSD_3_clause + file LICENSE

**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** jsonlite, rlang, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Config/roxygen2/version** 8.1.1

**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** 2026-11-02 21:40:02 UTC

## Contents

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

abort . . . 6
ansi256_to_hex . . . 7
assert_config_contract . . . 8
assert_description_language . . . 9
bold . . . 10
character_scalar . . . 11
checkmark . . . 11
check_bounded_double_scalar . . . 12
check_bounded_integer_scalar . . . 13
check_character . . . 14
check_character_scalar . . . 15
check_data.table . . . 16
check_dependencies . . . 16
check_double_scalar . . . 17
check_enum . . . 18
check_file_exists . . . 18
check_float01exc . . . 19
check_float01inc . . . 20
check_float0pos . . . 20
check_floatpos . . . 21
check_floatpos1 . . . 22
check_float_neg1_1 . . . 23
check_inherits . . . 23
check_integer_scalar . . . 24
check_logical . . . 25
check_logical_scalar . . . 26
check_nonneg_double_scalar . . . 26
check_nonneg_double_vector . . . 27
check_numeric . . . 28
check_optional_bounded_double_scalar . . . 29
check_optional_bounded_integer_scalar . . . 30
check_optional_character_scalar . . . 31
check_optional_double_scalar . . . 31
check_optional_integer_scalar . . . 32
check_optional_logical_scalar . . . 33
check_optional_nonneg_double_scalar . . . 34
check_optional_nonneg_double_vector . . . 34
check_optional_pos_double_scalar . . . 35
check_optional_pos_double_vector . . . 36
check_optional_pos_integer_scalar . . . 37
check_optional_prob_scalar . . . 37
check_optional_prob_vector . . . 38
check_optional_scalar_character . . . 39
check_optional_unit_open_vector . . . 40
check_pos_double_scalar . . . 40
check_pos_double_vector . . . 41
check_pos_integer_scalar . . . 42
check_prob_scalar . . . 43

check_prob_vector . . . 43
check_scalar_character . . . 44
check_scalar_logical . . . 45
check_tabular . . . 46
check_unit_open_scalar . . . 46
check_unit_open_vector . . . 47
clean_colnames . . . 48
clean_int . . . 48
clean_names . . . 49
clean_posint . . . 50
col256 . . . 51
collapse_head . . . 51
crossmark . . . 52
dbg . . . 53
ddSci . . . 54
double_scalar . . . 55
enum . . . 55
fmt . . . 56
fmt_gradient . . . 57
format_trace . . . 58
get_msg_sink . . . 59
get_output_type . . . 60
get_verbosity . . . 61
highlight . . . 61
html_div . . . 62
html_escape . . . 63
html_li . . . 63
html_p . . . 64
html_raw . . . 65
html_span . . . 65
html_strong . . . 66
html_tag . . . 67
html_ul . . . 68
info . . . 68
integer_scalar . . . 69
labelify . . . 70
logical_scalar . . . 71
match_arg . . . 71
msg . . . 72
msgdone . . . 73
msgstart . . . 74
nonneg_double_scalar . . . 75
nonneg_double_vector . . . 75
nonneg_integer_scalar . . . 76
optional . . . 77
optional_character_scalar . . . 77
optional_double_scalar . . . 78
optional_integer_scalar . . . 79

optional_logical_scalar . . . 79
optional_nonneg_double_scalar . . . 80
optional_nonneg_double_vector . . . 80
optional_nonneg_integer_scalar . . . 81
optional_pos_double_scalar . . . 82
optional_pos_double_vector . . . 82
optional_pos_integer_scalar . . . 83
optional_prob_scalar . . . 84
optional_prob_vector . . . 84
optional_unit_open_scalar . . . 85
optional_unit_open_vector . . . 86
plain . . . 86
pos_double_scalar . . . 87
pos_double_vector . . . 88
pos_integer_scalar . . . 88
printdf . . . 89
printls . . . 90
prob_scalar . . . 92
prob_vector . . . 92
progress_begin . . . 93
progress_clear . . . 94
progress_end . . . 95
progress_lapply . . . 96
progress_update . . . 98
prop_bag . . . 99
prop_boolean . . . 99
prop_const . . . 100
prop_float . . . 101
prop_integer . . . 102
prop_spec . . . 103
prop_string . . . 104
repr . . . 105
repr_ls . . . 106
rtemis_colors . . . 107
set_msg_sink . . . 108
strip_ansi . . . 109
success . . . 110
test_inherits . . . 110
unit_open_scalar . . . 111
unit_open_vector . . . 112
warn . . . 112
with_msg_sink . . . 113
write_JSONSchema . . . 114
write_lines . . . 115

**Index** **116**

## 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 *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)
```

---

abort *Dual-channel error signal*

---

## 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,
  data = NULL,
  parent = NULL,
  verbosity = NULL,
  package = NULL
)
```

## Arguments

| | |
|---|---|
| `...` | Message components, concatenated with no separator. |
| `class` | Character vector: Additional condition classes (prepended to the base `c("rtemis_error", "error", "condition")`). |
| `data` | Named list or NULL: Structured fields attached to the signalled condition, retrievable by handlers via `condition$<name>` (e.g. `data = list(status_code = 429L, provider = "anthropic")`). Fields ride on the condition only - they are not echoed to the console or appended to the message. Names must not collide with the built-in condition fields `message`, `parent`, `call`, `calls`. `trace` is rejected too - not because `abort()` sets it, but because rlang and testthat read that field as an `rlang::trace_back()` object (see details). |
| `parent` | Condition or NULL: Wrapped parent condition. Its message is echoed to the console (when verbosity allows) and stored on the signalled condition as `$parent`. |
| `verbosity` | Integer or NULL: Overrides `get_verbosity()` for the console echo. Set to `0L` 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 `$calls` - a `pairlist` of `sys.calls()` captured at the abort site, with `abort()`'s own frame trimmed. Unlike base R's `traceback()` (which only sees `.Traceback`, populated only when an error reaches the top-level uncaught), `$calls` 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.

The field is deliberately not named `trace`: rlang and testthat treat that name as an `rlang::trace_back()` object and call `nrow()` on it, so a `pairlist` there makes `testthat`'s reporter fail while formatting the error. Leaving the name unclaimed also lets testthat show its own backtrace, which is pruned to user frames where this one is not. For the same reason `abort()` rejects `data$trace`, so a caller cannot put the crash back by another route.

## 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` *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)
```

---

## assert_config_contract

*Assert a generated config schema honors the input-schema contract*

---

### Description

Checks one generated JSON Schema against the rules a *config* document must obey, and throws if any is broken. Shared by every package that publishes to schema.rtemis.org – rtemis and rtemis.draw – so one registry cannot hold documents held to two standards.

### Usage

```
assert_config_contract(schema, id = schema[["$id"]], structural = character())
```

### Arguments

| | |
|---|---|
| schema | Named list: The generated schema, as `S7_to_JSONSchema()` or `S7_dispatcher_JSONSchema()` returns it. |
| id | Character: The schema's `$id`, used to name it in the error. |
| structural | Character: Keys this schema may require because they carry the document's shape rather than a value – a family dispatcher's discriminator. Empty for a leaf or a flat config. |

### Details

Five rules, each recorded where it is raised:

- No top-level `required` beyond the key carrying the document's shape: a family dispatcher's discriminator, which selects the variant whose settings are its siblings. A config is otherwise a partial expression of intent.
- No `default`: defaults are versioned separately, in `defaults/v1`.

- No conditional demand for a key (`then`/`else` with `required`, or `dependentRequired`): an implementation could satisfy it by filling a value, which makes it a resolution rule and belongs to the record form.
- No R construct named in a description: the corpus is language-independent and is read by R, by the Rust CLI, by the browser, and by a model that writes no code at all.
- No R spelling of a value in a description: `NULL`, `TRUE`, `FALSE`, `NA` and the `@property` accessor are R, not JSON.

The first three are about what a config may demand, and record schemas are not subject to them: a record states what a run used, so everything in it is required. The last two are about prose, which every published document has; `assert_description_language()` applies just those, for a record or a result class that does not come through here.

## Value

The `schema`, invisibly, so it can wrap a write call. Throws with class `simpleError` listing every rule broken, so one run reports all of them rather than the first.

## Author(s)

EDG

## Examples

```
assert_config_contract(
  list(type = "object", properties = list(k = list(type = "integer"))),
  "https://schema.rtemis.org/example/v1/schema.json"
)
```

---

# assert_description_language

*Assert a published schema's descriptions are language-independent*

---

## Description

The two prose rules of the input-schema contract, on their own: no description may name an R construct, and none may spell a value the way R does. `assert_config_contract()` applies both along with the rules about what a config may demand; this is the entry point for a document those do not govern – a record, or a result class whose `required` states what rtemis always writes.

## Usage

```
assert_description_language(schema, id = schema[["$id"]])
```

## Arguments

| | |
|---|---|
| `schema` | Named list: The generated schema, as `S7_to_JSONSchema()` or `S7_dispatcher_JSONSchema()` returns it. |
| `id` | Character: The schema's `$id`, used to name it in the error. |

## Details

Every published document is read by R, by the Rust CLI, by the browser and by a model that writes no code at all, so its prose is part of the interface rather than a comment on it. A description reading "NULL = unweighted" is correct roxygen and an invalid instruction to every reader but one, who must write `null`; the fix is to say what the absent value means, not to transliterate the literal.

## Value

The `schema`, invisibly, so it can wrap a write call. Throws with class `simpleError` naming every offending description, so one run reports all of them rather than the first.

## Author(s)

EDG

## Examples

```
assert_description_language(
  list(
    type = "object",
    properties = list(k = list(type = "integer", description = "Clusters."))
  ),
  "https://schema.rtemis.org/example/v1/schema.json"
)
```

---

`bold` *Make text bold*

---

## Description

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

## Usage

```
bold(text, reset_code = "\033[22m", output_type = NULL)
```

## Arguments

| | |
|---|---|
| `text` | Character: Text to make bold. |
| `reset_code` | Character: ANSI reset code to use after formatting. |
| `output_type` | Character ("ansi", "html", "plain") or NULL: Output type. If NULL, resolved via `get_output_type()`. |

## Value

Character: Formatted text with bold styling

### Author(s)

EDG

### Examples

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

---

## character_scalar *Non-empty character scalar S7 property*

---

### 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

### Examples

```
Model <- S7::new_class("Model", properties = list(algorithm = character_scalar))
Model(algorithm = "LightGBM")@algorithm
try(Model(algorithm = ""))
```

---

## checkmark *Checkmark*

---

### Description

Prints a checkmark symbol with optional color and formatting.

### Usage

```
checkmark(col = col_success, output_type = NULL)
```

### Arguments

| | |
|---|---|
| col | Color for the checkmark symbol. |
| output_type | Character ("ansi", "html", "plain") or NULL: Output type. If NULL, resolved via get_output_type(). |

### Value

Character: Formatted checkmark symbol.

### Author(s)

EDG

### Examples

```
checkmark()
```

---

## check_bounded_double_scalar

*Check double scalar within bounds*

---

### Description

Check double scalar within bounds

### Usage

```
check_bounded_double_scalar(
  x,
  lower = -Inf,
  upper = Inf,
  lower_open = FALSE,
  upper_open = FALSE,
  arg_name = deparse(substitute(x))
)
```

### Arguments

| | |
|---|---|
| x | Numeric: Value to check. Must be a single finite non-NA number. |
| lower | Numeric scalar: Lower bound. |
| upper | Numeric scalar: Upper bound. |
| lower_open | Logical scalar: If TRUE, the lower bound is exclusive. |
| upper_open | Logical scalar: If TRUE, the upper bound is exclusive. |
| arg_name | Character: Argument name to use in error messages. |

### Details

For bounds not covered by the fixed-range checks (check_prob_scalar is `[0, 1]`, check_pos_double_scalar is `(0, Inf)`), or where the range is not known until runtime. The argument-checking counterpart to prop_float, whose `min`/`max` and `exclusive_min`/`exclusive_max` describe the same intervals for a config class. Finite like it: `Inf`/`-Inf` are rejected even where the matching bound is unbounded, since an infinite bound is always an open one.

### Value

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

### Author(s)

EDG

### Examples

```
check_bounded_double_scalar(0.7, lower = 0, upper = 2)
# Learning rate in (0, 1]
check_bounded_double_scalar(0.1, lower = 0, upper = 1, lower_open = TRUE)
# Throw error:
try(check_bounded_double_scalar(5, lower = 0, upper = 2))
try(check_bounded_double_scalar(Inf, lower = 0))
```

---

## check_bounded_integer_scalar

*Check integer scalar within bounds*

---

### Description

Check integer scalar within bounds

### Usage

```
check_bounded_integer_scalar(
  x,
  lower = -Inf,
  upper = Inf,
  arg_name = deparse(substitute(x))
)
```

### Arguments

| | |
|---|---|
| `x` | Numeric: Value to check. Must be a single non-NA whole number. |
| `lower` | Numeric scalar: Lower bound, inclusive. |
| `upper` | Numeric scalar: Upper bound, inclusive. |
| `arg_name` | Character: Argument name to use in error messages. |

## Details

For bounds not covered by the fixed-range checks: `check_pos_integer_scalar()` is `[1, Inf)`, and there is no non-negative or upper-bounded equivalent. Integer-typed inputs (`5L`) and double-typed whole numbers (`5`) are both accepted, matching check_integer_scalar. Both bounds are inclusive; an exclusive integer bound is the next whole number in.

## Value

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

## Author(s)

EDG

## Examples

```
check_bounded_integer_scalar(5L, lower = 0, upper = 20)
check_bounded_integer_scalar(0, lower = 0)
# Throw error:
try(check_bounded_integer_scalar(25, lower = 0, upper = 20))
try(check_bounded_integer_scalar(1.5, lower = 0))
```

---

# 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_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 *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_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 *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_file_exists *Check file exists*

### 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 *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_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

| | |
|---|---|
| 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_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` *Check numeric*

---

## 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

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

check_optional_bounded_double_scalar
*Check optional double scalar within bounds*

## Description

Check optional double scalar within bounds

## Usage

```
check_optional_bounded_double_scalar(
  x,
  lower = -Inf,
  upper = Inf,
  lower_open = FALSE,
  upper_open = FALSE,
  arg_name = deparse(substitute(x))
)
```

## Arguments

| | |
|---|---|
| x | Optional Numeric: Value to check. Must be NULL or a single finite non-NA number within bounds. |
| lower | Numeric scalar: Lower bound. |
| upper | Numeric scalar: Upper bound. |
| lower_open | Logical scalar: If TRUE, the lower bound is exclusive. |
| upper_open | Logical scalar: If TRUE, the upper bound is exclusive. |
| 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_bounded_double_scalar(NULL)
check_optional_bounded_double_scalar(0.7, lower = 0, upper = 2)
# Throw error:
try(check_optional_bounded_double_scalar(5, lower = 0, upper = 2))
```

check_optional_bounded_integer_scalar
*Check optional integer scalar within bounds*

## Description

Check optional integer scalar within bounds

## Usage

```
check_optional_bounded_integer_scalar(
  x,
  lower = -Inf,
  upper = Inf,
  arg_name = deparse(substitute(x))
)
```

## Arguments

| | |
|---|---|
| x | Optional Numeric: Value to check. Must be NULL or a single non-NA whole number within bounds. |
| lower | Numeric scalar: Lower bound, inclusive. |
| upper | Numeric scalar: Upper bound, inclusive. |
| 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_bounded_integer_scalar(NULL)
check_optional_bounded_integer_scalar(5L, lower = 0, upper = 20)
# Throw error:
try(check_optional_bounded_integer_scalar(25, lower = 0, upper = 20))
```

## 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 *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)))
```

## check_unit_open_scalar *Check open-unit-interval scalar*

### 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 *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, 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` *Check positive integer*

---

## 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 = NULL)
```

### 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 ("ansi", "html", "plain") or NULL: Output type. If NULL, resolved via get_output_type(). |

### 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
)
```

---

## crossmark *Cross mark*

---

### Description

Prints a cross mark symbol with optional color and formatting.

### Usage

```
crossmark(col = rtemis_colors[["red"]], output_type = NULL)
```

### Arguments

| | |
|---|---|
| col | Color for the cross mark symbol. |
| output_type | Character ("ansi", "html", "plain") or NULL: Output type. If NULL, resolved via get_output_type(). |

### Value

Character: Formatted cross mark symbol.

### Author(s)

EDG

### Examples

```
crossmark()
```

---

dbg *Debug log message*

---

### Description

Muted log message gated at verbosity `>= 2L`. Use for development / troubleshooting output that should not appear in normal operation.

### Usage

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

### Arguments

| | |
|---|---|
| `...` | Message components, concatenated with no separator. |
| `verbosity` | Integer or NULL: Overrides `get_verbosity()` when supplied. |
| `package` | 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

| | |
|---|---|
| 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"
```

## double_scalar *Double scalar S7 property*

### Description

S7 property accepting a single non-NA double value.

### Usage

```
double_scalar
```

### Value

An S7 property object.

### Author(s)

EDG

### Examples

```
Metric <- S7::new_class("Metric", properties = list(value = double_scalar))
Metric(value = -1.5)@value
try(Metric(value = c(1, 2)))
```

## enum *Create an enum S7 property*

### 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 = NULL
)
```

### 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 ("ansi", "html", "plain") or NULL: Output type. If NULL, resolved via get_output_type(). |

### 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 = NULL)
```

### 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 ("ansi", "html", "plain") or NULL: Output type. If NULL, resolved via get_output_type(). |

### 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 `$calls` 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

| | |
|---|---|
| `trace` | `pairlist` of calls, as captured by `abort()` on `cond$calls`. Passing the condition itself also works - the calls are extracted via `cond[["calls"]]`, falling back to an rlang trace on `cond[["trace"]]`. |
| `max_width` | Integer: Max characters per deparsed line. Longer calls are truncated with a trailing ellipsis. Ignored for rlang traces, which rlang formats itself. |

### Details

Conditions from packages built on `rlang::abort()` carry their stack as an rlang trace object on `$trace` rather than on `$calls`. Those are passed to rlang's own formatter, so the result is rlang's tree layout and `max_width` does not apply; any ANSI styling is stripped.

### 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` *Get the current rtemis message sink*

---

## 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()`.

## Examples

```
get_msg_sink() # NULL unless a sink is set
old <- set_msg_sink(function(m) invisible(m))
is.function(get_msg_sink())
set_msg_sink(old) # restore the previous sink
```

`get_output_type` *Get output type*

## Description

Resolve the output type for printing text.

## Usage

```
get_output_type(output_type = NULL, filename = NULL)
```

## Arguments

| | |
|---|---|
| `output_type` | Character ("ansi", "html", "plain") or NULL: Output type. If NULL, resolved by environment, see Details. |
| `filename` | Optional Character: Filename for output. If not NULL, forces "plain". |

## Details

Exported as internal function for use by other rtemis packages. `fmt()` and its wrappers resolve their `output_type` argument through this function, so callers that only forward `output_type` to formatting functions do not need to call it themselves. Call it directly when the resolved value drives logic other than text formatting (e.g. whether to draw a spinner) or when output may be redirected to a file.

When `output_type` is NULL, the type is resolved in this order, first match winning:

1. `filename` is not NULL: "plain".
2. `getOption("rtemis.output_type")`, e.g. `options(rtemis.output_type = "ansi")` in `.Rprofile`.
3. The `RTEMIS_OUTPUT_TYPE` environment variable, which lets a parent process (the `rtemis` CLI, a scheduler job script) decide per invocation.
4. `NO_COLOR` set to a non-empty value (https://no-color.org): "plain".
5. "ansi" in interactive sessions, "plain" otherwise.

Unrecognized values in 2 and 3 are ignored rather than raising, so a typo falls through to the next rule.

With "ansi" output, `progress_begin()` and friends render a carriage-return-rewritten status line in interactive sessions outside knitting. Noninteractive and knitted runs retain the final completion summary. Non-"ansi" output uses `msg0()` start and completion lines.

## Value

Character with selected output type.

## Author(s)

EDG

## Examples

```
get_output_type()
```

---

`get_verbosity` *Resolve the current logging verbosity*

---

## Description

Reads `getOption("<package>.verbosity")` first when `package` is supplied, falling back to `getOption("rtemis.verbosi` 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` *Highlight text*

---

## Description

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

## Usage

```
highlight(x, pad = 0L, output_type = NULL)
```

### Arguments

| | |
|---|---|
| x | Character: Text to highlight. |
| pad | Integer: Number of spaces to pad before text. |
| output_type | Character ("ansi", "html", "plain") or NULL: Output type. If NULL, resolved via get_output_type(). |

### Value

Character: Formatted text with highlight.

### Author(s)

EDG

### Examples

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

---

## html_div *Build a* div *element*

---

### Description

Build a div element

### Usage

```
html_div(..., class = NULL, style = NULL, id = NULL)
```

### Arguments

| | |
|---|---|
| ... | Children: Character, numbers, rtemis_html objects, or lists of those. |
| class | Optional Character: class attribute. Must be NULL or a single string; collapse multiple classes yourself, e.g. paste(x, collapse = " "). |
| style | Optional Character: style attribute. Must be NULL or a single string. |
| id | Optional Character: id attribute. Must be NULL or a single string. |

### Value

Character of class rtemis_html_element and rtemis_html.

### Author(s)

EDG

### Examples

```
html_div("Contents", class = "panel")
```

## html_escape *Escape text for inclusion in HTML*

### Description

Escape text for inclusion in HTML

### Usage

```
html_escape(x, attribute = FALSE)
```

### Arguments

| | |
|---|---|
| x | Character: Text to escape. Coerced with `as.character()`. |
| attribute | Logical: If TRUE, also escape the quote characters, which must be escaped inside an attribute value but not in element text. Must be a single non-NA TRUE or FALSE. |

### Value

Character: Escaped text, carrying no class.

### Author(s)

EDG

### Examples

```
html_escape("a < b & c")
html_escape('say "hi"', attribute = TRUE)
```

## html_li *Build a* `li` *element*

### Description

Build a `li` element

### Usage

```
html_li(..., class = NULL, style = NULL, id = NULL)
```

### Arguments

| | |
|---|---|
| ... | Children: Character, numbers, rtemis_html objects, or lists of those. |
| class | Optional Character: class attribute. Must be NULL or a single string; collapse multiple classes yourself, e.g. paste(x, collapse = " "). |
| style | Optional Character: style attribute. Must be NULL or a single string. |
| id | Optional Character: id attribute. Must be NULL or a single string. |

### Value

Character of class rtemis_html_element and rtemis_html.

### Author(s)

EDG

### Examples

```
html_li("An item")
```

---

## html_p *Build a* p *element*

---

### Description

Build a p element

### Usage

```
html_p(..., class = NULL, style = NULL, id = NULL)
```

### Arguments

| | |
|---|---|
| ... | Children: Character, numbers, rtemis_html objects, or lists of those. |
| class | Optional Character: class attribute. Must be NULL or a single string; collapse multiple classes yourself, e.g. paste(x, collapse = " "). |
| style | Optional Character: style attribute. Must be NULL or a single string. |
| id | Optional Character: id attribute. Must be NULL or a single string. |

### Value

Character of class rtemis_html_element and rtemis_html.

### Author(s)

EDG

### Examples

```
html_p("A paragraph.")
```

---

`html_raw` *Mark a string as HTML*

---

### Description

Declares that `x` is already markup, so it is embedded verbatim rather than escaped when it becomes the child of an element. The counterpart to `html_escape()`: use it for markup you assembled yourself, never for text that came from outside.

### Usage

```
html_raw(x)
```

### Arguments

| | |
|---|---|
| `x` | Character: Markup. |

### Value

Character of class `rtemis_html`.

### Author(s)

EDG

### Examples

```
html_span(html_raw("<em>emphasis</em>"))
# Without the marker the same string is text, and escaped:
html_span("<em>emphasis</em>")
```

---

`html_span` *Build a* `span` *element*

---

### Description

Build a `span` element

### Usage

```
html_span(..., class = NULL, style = NULL, id = NULL)
```

### Arguments

| | |
|---|---|
| ... | Children: Character, numbers, rtemis_html objects, or lists of those. |
| class | Optional Character: class attribute. Must be NULL or a single string; collapse multiple classes yourself, e.g. paste(x, collapse = " "). |
| style | Optional Character: style attribute. Must be NULL or a single string. |
| id | Optional Character: id attribute. Must be NULL or a single string. |

### Value

Character of class rtemis_html_element and rtemis_html.

### Author(s)

EDG

### Examples

```
html_span("Inline text", style = "color: #16A0AC;")
```

---

## html_strong *Build a* strong *element*

---

### Description

Build a strong element

### Usage

```
html_strong(..., class = NULL, style = NULL, id = NULL)
```

### Arguments

| | |
|---|---|
| ... | Children: Character, numbers, rtemis_html objects, or lists of those. |
| class | Optional Character: class attribute. Must be NULL or a single string; collapse multiple classes yourself, e.g. paste(x, collapse = " "). |
| style | Optional Character: style attribute. Must be NULL or a single string. |
| id | Optional Character: id attribute. Must be NULL or a single string. |

### Value

Character of class rtemis_html_element and rtemis_html.

### Author(s)

EDG

## Examples

```
html_strong(42L)
```

---

`html_tag` *Build an HTML element*

---

## Description

Children given as `...` are escaped unless marked with `html_raw()` or produced by another `html_*` constructor. Lists are flattened, so children can be built with `lapply()`, and NULL children are dropped.

## Usage

```
html_tag(name, ..., class = NULL, style = NULL, id = NULL)
```

## Arguments

| | |
|---|---|
| `name` | Character: Element name, e.g. "div". Must be a single valid element name: a letter followed by letters, digits, or hyphens. |
| `...` | Children: Character, numbers, `rtemis_html` objects, or lists of those. |
| `class` | Optional Character: `class` attribute. Must be NULL or a single string; collapse multiple classes yourself, e.g. `paste(x, collapse = " ")`. |
| `style` | Optional Character: `style` attribute. Must be NULL or a single string. |
| `id` | Optional Character: `id` attribute. Must be NULL or a single string. |

## Details

A tag holding a single text child renders on one line; anything else renders as an indented block, and a child's own line breaks are indented with it, so indentation always tracks nesting depth.

## Value

Character of class `rtemis_html_element` and `rtemis_html`.

## Author(s)

EDG

## Examples

```
html_tag("section", "Body text", class = "intro")
html_tag("ul", lapply(c("one", "two"), html_li))
```

## html_ul *Build a* ul *element*

### Description

Build a ul element

### Usage

```
html_ul(..., class = NULL, style = NULL, id = NULL)
```

### Arguments

| | |
|---|---|
| ... | Children: Character, numbers, rtemis_html objects, or lists of those. |
| class | Optional Character: class attribute. Must be NULL or a single string; collapse multiple classes yourself, e.g. paste(x, collapse = " "). |
| style | Optional Character: style attribute. Must be NULL or a single string. |
| id | Optional Character: id attribute. Must be NULL or a single string. |

### Value

Character of class rtemis_html_element and rtemis_html.

### Author(s)

EDG

### Examples

```
html_ul(html_li("one"), html_li("two"))
```

## info *Informational log message*

### 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

| | |
|---|---|
| `...` | Message components, concatenated with no separator. |
| `verbosity` | Integer or NULL: Overrides `get_verbosity()` when supplied. |
| `package` | 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` *Integer scalar S7 property*

---

### 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

### Examples

```
Params <- S7::new_class("Params", properties = list(seed = integer_scalar))
Params(seed = 42L)@seed
# Doubles are not accepted: the type must be integer
try(Params(seed = 42))
```

`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)
```

---

`logical_scalar` *Logical scalar S7 property*

---

## Description

S7 property accepting a single non-NA logical value.

## Usage

```
logical_scalar
```

## Value

An S7 property object.

## Author(s)

EDG

## Examples

```
Params <- S7::new_class("Params", properties = list(scale = logical_scalar))
Params(scale = TRUE)@scale
try(Params(scale = NA))
```

---

`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 ... |

### Details

Avoid `msgstart()`/`msgdone()` pairs that span `progress_update()` calls: a progress redraw closes the pending line, so the checkmark printed by `msgdone()` lands on a fresh line instead of completing the original one. Prefer `msg()` for messages emitted inside progress loops.

### 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

### Examples

```
Penalty <- S7::new_class("Penalty", properties = list(alpha = nonneg_double_scalar))
Penalty(alpha = 0)@alpha
try(Penalty(alpha = -1))
```

---

## 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

## Examples

```
Importance <- S7::new_class(
  "Importance",
  properties = list(scores = nonneg_double_vector)
)
Importance(scores = c(0, 1.5, 3))@scores
try(Importance(scores = c(1, -1)))
```

---

# 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

## Examples

```
Counter <- S7::new_class(
  "Counter",
  properties = list(n_failed = nonneg_integer_scalar)
)
Counter(n_failed = 0L)@n_failed
try(Counter(n_failed = -1L))
```

## 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

## Examples

```
Model <- S7::new_class("Model", properties = list(label = optional_character_scalar))
Model()@label
Model(label = "Experiment 1")@label
try(Model(label = ""))
```

---

`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

## Examples

```
Metric <- S7::new_class("Metric", properties = list(offset = optional_double_scalar))
Metric()@offset
Metric(offset = 0.5)@offset
try(Metric(offset = NA_real_))
```

## 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

### Examples

```
Params <- S7::new_class("Params", properties = list(seed = optional_integer_scalar))
Params()@seed
Params(seed = 42L)@seed
try(Params(seed = 42))
```

## 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

## Examples

```
Params <- S7::new_class("Params", properties = list(scale = optional_logical_scalar))
Params()@scale
Params(scale = FALSE)@scale
try(Params(scale = c(TRUE, FALSE)))
```

---

# 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

## Examples

```
Penalty <- S7::new_class(
  "Penalty",
  properties = list(alpha = optional_nonneg_double_scalar)
)
Penalty()@alpha
Penalty(alpha = 0)@alpha
try(Penalty(alpha = -1))
```

---

# 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

## Examples

```
Importance <- S7::new_class(
  "Importance",
  properties = list(scores = optional_nonneg_double_vector)
)
Importance()@scores
Importance(scores = c(0, 2))@scores
try(Importance(scores = c(0, -2)))
```

---

# 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

## Examples

```
Counter <- S7::new_class(
  "Counter",
  properties = list(n_failed = optional_nonneg_integer_scalar)
)
Counter()@n_failed
Counter(n_failed = 3L)@n_failed
try(Counter(n_failed = -1L))
```

## 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

### Examples

```
Penalty <- S7::new_class(
  "Penalty",
  properties = list(gamma = optional_pos_double_scalar)
)
Penalty()@gamma
Penalty(gamma = 2)@gamma
try(Penalty(gamma = Inf))
```

## 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

### Examples

```
Grid <- S7::new_class("Grid", properties = list(lambdas = optional_pos_double_vector))
Grid()@lambdas
Grid(lambdas = c(0.01, 0.1, 1))@lambdas
try(Grid(lambdas = c(0.1, Inf)))
```

---

## 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

### Examples

```
Resample <- S7::new_class(
  "Resample",
  properties = list(n_workers = optional_pos_integer_scalar)
)
Resample()@n_workers
Resample(n_workers = 4L)@n_workers
try(Resample(n_workers = 0L))
```

## 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

### Examples

```
Split <- S7::new_class("Split", properties = list(threshold = optional_prob_scalar))
Split()@threshold
Split(threshold = 0.5)@threshold
try(Split(threshold = -0.1))
```

## 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

## Examples

```
Preds <- S7::new_class(
  "Preds",
  properties = list(probabilities = optional_prob_vector)
)
Preds()@probabilities
Preds(probabilities = c(0.1, 0.9))@probabilities
try(Preds(probabilities = c(0.1, 1.1)))
```

---

`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

## Examples

```
Boost <- S7::new_class(
  "Boost",
  properties = list(subsample = optional_unit_open_scalar)
)
Boost()@subsample
Boost(subsample = 0.8)@subsample
try(Boost(subsample = 0))
```

## 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

### Examples

```
Grid <- S7::new_class(
  "Grid",
  properties = list(subsamples = optional_unit_open_vector)
)
Grid()@subsamples
Grid(subsamples = c(0.5, 0.8))@subsamples
try(Grid(subsamples = c(0.5, 1)))
```

## 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` *Positive double scalar S7 property*

---

## 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

## Examples

```
Penalty <- S7::new_class("Penalty", properties = list(lambda = pos_double_scalar))
Penalty(lambda = 0.01)@lambda
try(Penalty(lambda = 0))
```

## pos_double_vector *Positive double vector S7 property*

### 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

### Examples

```
Weights <- S7::new_class(
  "Weights",
  properties = list(case_weights = pos_double_vector)
)
Weights(case_weights = c(0.5, 1, 2))@case_weights
try(Weights(case_weights = c(1, 0)))
```

## pos_integer_scalar *Positive integer scalar S7 property*

### 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

## Examples

```
Resample <- S7::new_class(
  "Resample",
  properties = list(n_resamples = pos_integer_scalar)
)
Resample(n_resamples = 10L)@n_resamples
try(Resample(n_resamples = 0L))
```

---

`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 collapse_head() 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

### Examples

```
Split <- S7::new_class("Split", properties = list(train_p = prob_scalar))
Split(train_p = 0.75)@train_p
try(Split(train_p = 1.5))
```

## 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

### Examples

```
Preds <- S7::new_class("Preds", properties = list(probabilities = prob_vector))
Preds(probabilities = c(0, 0.5, 1))@probabilities
try(Preds(probabilities = c(0.5, NA)))
```

progress_begin *Begin a progress node*

## Description

Creates a progress handle, pushes it on the active-progress stack (nesting under the current innermost handle, if any), emits a `status = "start"` sink event when a message sink is set (see `set_msg_sink()`), and renders the console status line otherwise.

## Usage

```
progress_begin(
  total,
  label = "Progress",
  kind = "progress",
  id = NULL,
  parent_id = NULL,
  verbosity = NULL,
  package = NULL,
  output_type = NULL
)
```

## Arguments

| | |
|---|---|
| `total` | Integer: Total number of steps, or `NA` for indeterminate progress. Must be non-negative. |
| `label` | Character: Display label for this progress node. |
| `kind` | Character: Node kind forwarded in the sink envelope (e.g. `"tune"`, `"grid_cell"`). |
| `id` | Character or NULL: Node id for the sink envelope. Auto-generated (`"pb1"`, `"pb2"`, ...) when NULL; pass an execution-graph node id to correlate progress with other node events. |
| `parent_id` | Character or NULL: Parent id reported in the sink envelope when this handle is the outermost one on the stack. Lets a caller graft its progress nodes onto an enclosing execution graph (e.g. rtemis passes the node its loop is running inside), so a consumer reading both streams reconstructs one tree. Ignored when the handle nests inside another handle, whose id is then the parent. |
| `verbosity` | Integer or NULL: Overrides `get_verbosity()` when supplied. Gates only the console renderer, never sink events. |
| `package` | Character or NULL: Package name for verbosity override lookup (e.g. `"rtemis"`). |
| `output_type` | Character or NULL: `"ansi"`, `"html"`, or `"plain"`; resolved via `get_output_type()` when NULL (`"ansi"` in interactive sessions, `"plain"` otherwise). Animation requires `"ansi"` output in an interactive session outside knitting. |

## Details

The console renderer writes a single status line rewritten in place, with a color-pulsing spinner and a breadcrumb of all active levels, e.g. `Outer resamples 2/5 > Tuning 7/30 ETA 0:41`. Each level shows the step currently in flight (1-based), so a loop counts `1/n` through `n/n`; the ETA is computed from completed steps. Animation requires an interactive session with `getOption("knitr.in.progress")` other than `TRUE`. With `"ansi"` output, noninteractive and knitted runs retain the outermost completion line. Non-`"ansi"` output gets one begin line and one completion line per handle. Two options control rendering:

- `rtemis.progress_throttle`: Minimum seconds between redraws and between sink `"update"` events (default `0.1`). Begin/end events always fire; set to `0` to emit every tick.
- `rtemis.progress_spinner`: Spinner design - one of `"dots"` (braille dots, default), `"dot"` (static dot, color-only animation), or `"blocks"` (quadrant blocks). All designs share the palette color pulse (`light_orange` to `red`).

Sink events fire regardless of verbosity; verbosity gates only the console renderer.

**Known limitation**: output written directly to stdout by code running inside a progress loop (`cat()`, `print()`, verbose third-party routines) is not intercepted and will collide with the status line. Output through `msg()` and the other rtemis writers clears the line automatically, and `progress_lapply()` additionally clears it for any `message()`/`warning()` condition raised by user code. For raw stdout writes, call `progress_clear()` first or silence the routine (`capture.output()`, `verbose = 0`).

## Value

An `rtemis_progress` handle (environment), invisibly. Pass it to `progress_update()` and `progress_end()`.

## Author(s)

EDG

## See Also

Other progress: `progress_clear()`, `progress_end()`, `progress_lapply()`, `progress_update()`

## Examples

```
h <- progress_begin(3L, label = "Fitting", output_type = "plain")
progress_update(h)
progress_end(h)
```

---

# progress_clear *Clear the visible progress status line*

---

## Description

Escape hatch for code that is about to write raw console output (e.g. `cat()`, `print()`, or a verbose third-party routine) while a progress status line is on screen: clears the line so the output starts clean. The status line reappears on the next `progress_update()`. No-op when nothing is displayed.

### Usage

```
progress_clear()
```

### Details

Not needed for output that goes through `msg()` and the other rtemis writers (they clear automatically), nor for `message()`/`warning()` conditions raised inside `progress_lapply()` (a calling handler clears for those). This exists for the one gap that cannot be intercepted: direct writes to stdout.

### Value

NULL invisibly.

### Author(s)

EDG

### See Also

Other progress: `progress_begin()`, `progress_end()`, `progress_lapply()`, `progress_update()`

### Examples

```
progress_clear() # no-op when no status line is visible
```

---

## progress_end *End a progress node*

---

### Description

Closes the handle, pops it off the active-progress stack, and emits a final sink event or console output. Ending a handle that is not the innermost auto-closes everything nested above it as `"aborted"` (their sink events fire), so the stack stays consistent. No-op on an already closed handle.

### Usage

```
progress_end(handle, status = c("done", "error", "aborted"))
```

### Arguments

| | |
|---|---|
| `handle` | `rtemis_progress` handle from `progress_begin()`. |
| `status` | Character: `"done"`, `"error"`, or `"aborted"`. |

## Details

In "ansi" mode, including noninteractive and knitted runs, only the outermost end prints a permanent completion line (label n/total done in 0:41). Inner-level ends redraw the remaining breadcrumb in interactive sessions outside knitting. Non-"ansi" output prints a completion line per handle.

When nested loops complete uniformly, the completion line reports them as a multiplication chain, e.g. Outer 2/2 x Tuning 24/24 done in 0:41. A nested level is included only when every one of its runs completed with status = "done", reached its total, and had the same label and total as its sibling runs; otherwise the chain is omitted rather than reporting a misleading count. Deeper nesting chains recursively.

## Value

NULL invisibly.

## Author(s)

EDG

## See Also

Other progress: progress_begin(), progress_clear(), progress_lapply(), progress_update()

## Examples

```
h <- progress_begin(2L, label = "Fitting", output_type = "plain")
progress_update(h)
progress_end(h)
```

---

# progress_lapply lapply *with progress*

---

## Description

Drop-in replacement for lapply() that reports progress after each element: begins a progress node, ticks once per element, and ends the node. Nested calls produce a nested breadcrumb on the console (and parent_id-chained sink events). If fn throws, the node is ended with status = "error" before the condition propagates, so nested wrappers unwind cleanly.

## Usage

```
progress_lapply(
  X,
  FUN,
  ...,
  label = "Processing",
  kind = "progress",
```

```
  parent_id = NULL,
  verbosity = NULL,
  package = NULL,
  output_type = NULL
)
```

## Arguments

| | |
|---|---|
| X | Vector or list: Elements to iterate over, as in lapply(). |
| FUN | Function: Applied to each element of X. Named X/FUN to match lapply(), so arguments forwarded through ... (commonly x) never collide with the wrapper's own parameters. |
| ... | Additional arguments passed to FUN. |
| label | Character: Display label for the progress node. |
| kind | Character: Node kind forwarded in the sink envelope. |
| parent_id | Character or NULL: Parent id for the sink envelope when this is the outermost active handle - see progress_begin(). |
| verbosity | Integer or NULL: Overrides get_verbosity() when supplied. Gates only the console renderer, never sink events. |
| package | Character or NULL: Package name for verbosity override lookup. |
| output_type | Character or NULL: "ansi", "html", or "plain"; resolved via get_output_type() when NULL. |

## Details

message() and warning() conditions signalled by fn (or anything it calls) are intercepted by a calling handler that clears the status line before the text is printed, then lets normal handling proceed - so third-party verbose output lands on a clean line. Direct stdout writes (cat(), print()) cannot be intercepted; see progress_clear().

## Value

List: Exactly what lapply(X, FUN, ...) returns.

## Author(s)

EDG

## See Also

Other progress: progress_begin(), progress_clear(), progress_end(), progress_update()

## Examples

```
sqrts <- progress_lapply(
  1:4,
  sqrt,
  label = "Computing",
  output_type = "plain"
)
```

---

`progress_update` *Update a progress node*

---

## Description

Advances (or sets) the step counter of an active handle. Emits a throttled `status = "update"` sink event when a message sink is set, or redraws the console status line otherwise. No-op on a closed handle.

## Usage

```
progress_update(handle, current = NULL, add = 1L, label = NULL, force = FALSE)
```

## Arguments

| | |
|---|---|
| `handle` | `rtemis_progress` handle from `progress_begin()`. |
| `current` | Integer or NULL: Set the counter to this value. When NULL, the counter advances by add. Clamped to `[0, total]` for determinate handles. |
| `add` | Integer: Increment when `current` is NULL. |
| `label` | Character or NULL: Update the display label. |
| `force` | Logical: If TRUE, bypass the redraw/sink throttle (`getOption("rtemis.progress_throttle", 0.1)` seconds). |

## Value

The handle, invisibly.

## Author(s)

EDG

## See Also

Other progress: `progress_begin()`, `progress_clear()`, `progress_end()`, `progress_lapply()`

## Examples

```
h <- progress_begin(3L, label = "Fitting", output_type = "plain")
progress_update(h)
progress_update(h, current = 3L)
progress_end(h)
```

## prop_bag *Opaque named list S7 property*

### Description

A pass-through for values with no per-key contract of our own – extra request headers, backend-specific options – carried as one value however many keys it holds. The keys are required: the value becomes a JSON object, so every element must carry a name, and the names must be distinct. Use NULL, not `list()`, for no value.

### Usage

```
prop_bag(default = NULL, nullable = TRUE, description = "")
```

### Arguments

| | |
|---|---|
| `default` | List: Default value (NULL for none, or if `nullable`). Use a quoted call for a default that needs construction. The call is evaluated once in the caller's environment to validate the default and record it in `prop_spec()`, then again for each object in the class's environment. |
| `nullable` | Logical: If TRUE, NULL is a valid value. |
| `description` | Character: Human-readable description. |

### Value

S7 property.

### Author(s)

EDG

### Examples

```
extra_headers <- prop_bag(nullable = TRUE, description = "Extra headers")
```

## prop_boolean *Logical (boolean) S7 property*

### Description

Logical (boolean) S7 property

### Usage

```
prop_boolean(default = FALSE, nullable = FALSE, description = "")
```

### Arguments

| | |
|---|---|
| `default` | Logical: Default value (NULL only if `nullable`). Use a quoted call for a default that needs construction. The call is evaluated once in the caller's environment to validate the default and record it in `prop_spec()`, then again for each object in the class's environment. |
| `nullable` | Logical: If TRUE, NULL is a valid value. |
| `description` | Character: Human-readable description. |

### Value

S7 property.

### Author(s)

EDG

### Examples

```
verbose <- prop_boolean(default = TRUE, description = "Print progress")
```

---

## `prop_const` *Constant S7 property*

---

### Description

A property fixed to one value: it is the only value that validates, and it is the default, so the declaration is the whole contract.

### Usage

```
prop_const(value, description = "")
```

### Arguments

| | |
|---|---|
| `value` | Logical, numeric or character scalar: The value. |
| `description` | Character: Human-readable description. |

### Value

S7 property.

### Author(s)

EDG

### Examples

```
type <- prop_const("object", description = "JSON Schema type")
```

prop_float *Numeric (floating-point) S7 property*

## Description

The one factory whose name differs from its JSON Schema type: it emits type "number" (which in JSON Schema includes integers), but is named `prop_float` because declarers think in the integer/float pairing – "number" next to `prop_integer` invites the same ambiguity as R's "numeric". Accepts R integer values too (`class_numeric`), floats being a superset of integers.

## Usage

```
prop_float(
  default = NULL,
  min = NULL,
  max = NULL,
  exclusive_min = NULL,
  exclusive_max = NULL,
  nullable = FALSE,
  vector = FALSE,
  min_items = 1L,
  unique_items = FALSE,
  description = ""
)
```

## Arguments

| | |
|---|---|
| `default` | Numeric: Default value (NULL for none, or if `nullable`). Use a quoted call for a default that needs construction. The call is evaluated once in the caller's environment to validate the default and record it in `prop_spec()`, then again for each object in the class's environment. |
| `min`, `max` | Numeric or NULL: Inclusive bounds. |
| `exclusive_min`, `exclusive_max` | Numeric or NULL: Exclusive bounds. |
| `nullable` | Logical: If TRUE, NULL is a valid value. |
| `vector` | Logical: If TRUE, the value is vector-valued (a JSON array). |
| `min_items` | Integer: Fewest elements a `vector` value may hold. |
| `unique_items` | Logical: If TRUE, a `vector` value's elements must be distinct. |
| `description` | Character: Human-readable description. |

## Value

S7 property.

## Author(s)

EDG

## Examples

```
# Sampling temperature in [0, 2]
temperature <- prop_float(default = 0.3, min = 0, max = 2)
# Learning rate in (0, 1]
lr <- prop_float(default = 0.1, exclusive_min = 0, max = 1)
```

---

`prop_integer` *Integer S7 property*

---

## Description

Accepts R integers only (`3L`, not `3`), so that a whole-number contract is enforced by the type rather than by a rounding check.

## Usage

```
prop_integer(
  default = NULL,
  min = NULL,
  max = NULL,
  exclusive_min = NULL,
  exclusive_max = NULL,
  nullable = FALSE,
  vector = FALSE,
  min_items = 1L,
  unique_items = FALSE,
  description = ""
)
```

## Arguments

| | |
|---|---|
| `default` | Integer: Default value (NULL for none, or if `nullable`). Use a quoted call for a default that needs construction. The call is evaluated once in the caller's environment to validate the default and record it in `prop_spec()`, then again for each object in the class's environment. |
| `min`, `max` | Integer or NULL: Inclusive bounds. |
| `exclusive_min`, `exclusive_max` | Integer or NULL: Exclusive bounds. |
| `nullable` | Logical: If TRUE, NULL is a valid value. |
| `vector` | Logical: If TRUE, the value is vector-valued (a JSON array). |
| `min_items` | Integer: Fewest elements a `vector` value may hold. |
| `unique_items` | Logical: If TRUE, a `vector` value's elements must be distinct. |
| `description` | Character: Human-readable description. |

## Value

S7 property.

## Author(s)

EDG

## Examples

```
n_iter <- prop_integer(default = 100L, min = 1L, description = "Iterations")
```

---

`prop_spec` *A factory-built property's spec*

---

## Description

The machine-readable declaration behind a property: type, default, bounds, enum and description. Read it to generate documentation, a JSON Schema, or a defaults artifact from the class definition itself.

## Usage

```
prop_spec(property)
```

## Arguments

| | |
|---|---|
| `property` | S7 property, from one of the `prop_*` factories. |

## Value

Named list of spec fields, or NULL if the property was not built by a `prop_*` factory.

## Author(s)

EDG

## Examples

```
temperature <- prop_float(default = 0.3, min = 0, max = 2)
prop_spec(temperature)[["maximum"]]
```

prop_string *Character (string) S7 property*

## Description

Empty and whitespace-only strings are rejected by default: an unset value is `NULL`, so `""` reaching a property is almost always a mistake rather than a deliberate empty name. Pass `allow_empty = TRUE` where it is meaningful.

## Usage

```
prop_string(
  default = NULL,
  enum = NULL,
  nullable = FALSE,
  vector = FALSE,
  map = FALSE,
  min_items = 1L,
  unique_items = FALSE,
  allow_empty = FALSE,
  description = ""
)
```

## Arguments

| | |
|---|---|
| `default` | Character: Default value (NULL for none, or if `nullable`). Use a quoted call for a default that needs construction. The call is evaluated once in the caller's environment to validate the default and record it in `prop_spec()`, then again for each object in the class's environment. |
| `enum` | Character or NULL: Allowed values. |
| `nullable` | Logical: If TRUE, NULL is a valid value. |
| `vector` | Logical: If TRUE, the value is vector-valued (a JSON array). |
| `map` | Logical: If TRUE, the value is a *named* vector (a JSON object with string values). Mutually exclusive with `vector`. |
| `min_items` | Integer: Fewest elements a `vector` or `map` value may hold. |
| `unique_items` | Logical: If TRUE, a `vector` value's elements must be distinct. |
| `allow_empty` | Logical: If TRUE, `""` is a valid value. |
| `description` | Character: Human-readable description. |

## Value

S7 property.

## Author(s)

EDG

## Examples

```
model_name <- prop_string(description = "Model name")
backend <- prop_string(default = "ollama", enum = c("ollama", "openai"))
api_key <- prop_string(nullable = TRUE, description = "API key")
```

---

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 = NULL
)
```

## 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 ("ansi", "html", "plain") or NULL: Output type for mformat. If NULL, resolved via get_output_type(). |

## 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` *Set the rtemis message sink*

---

## 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`), `"done"` (`msgdone`), or `"progress"` (`progress_begin()` / `progress_update()` / `progress_end()`).

Producers may include **additional** fields in the list, which sinks should ignore when not understood. In particular, the progress API in this package and `rtemis`'s training observability emit 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"`, `"update"`, `"done"`, `"error"`, or `"aborted"`.
- `current`, `total`: integer or `NA`. Progress counters.
- `label`: character. Display label without the counts (e.g. `"Tuning"`). `text` is the label and counts already composed for display; a sink that renders its own layout wants this instead of parsing them apart.

These are additive; sinks that only read the base fields keep working. Progress events fire regardless of verbosity (verbosity gates only the console renderer), and `"update"` events are throttled via `getOption("rtemis.progress_throttle")` - see `progress_begin()`.

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 *Success log message*

### 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 get_verbosity() when supplied. |
| package | Character or NULL: Package name for verbosity override. |

### Value

Invisible NULL.

### Author(s)

EDG

### Examples

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

## 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)
```

---

## unit_open_scalar *Open-unit-interval scalar S7 property*

---

### Description

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

### Usage

```
unit_open_scalar
```

### Value

An S7 property object.

### Author(s)

EDG

### Examples

```
Boost <- S7::new_class("Boost", properties = list(learning_rate = unit_open_scalar))
Boost(learning_rate = 0.1)@learning_rate
# Bounds are exclusive: 0 and 1 are rejected
try(Boost(learning_rate = 1))
```

## unit_open_vector *Open-unit-interval vector S7 property*

### 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

### Examples

```
Grid <- S7::new_class("Grid", properties = list(learning_rates = unit_open_vector))
Grid(learning_rates = c(0.01, 0.1))@learning_rates
try(Grid(learning_rates = c(0.1, 1)))
```

## warn *Warning log message*

### 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, bold = FALSE, verbosity = NULL, package = NULL)
```

### Arguments

| | |
|---|---|
| `...` | Message components, concatenated with no separator. |
| `use_warning` | Logical: If TRUE, signal an R `warning` condition instead of (or in addition to) writing a styled message. |
| `bold` | Logical: If TRUE, apply bold styling to the message. |
| `verbosity` | Integer or NULL: Overrides `get_verbosity()` when supplied. |
| `package` | Character or NULL: Package name for verbosity override. |

## Value

Invisible NULL.

## Author(s)

EDG

## Examples

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

---

`with_msg_sink` *Run code with a temporary message sink*

---

## 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
  }
)
```

`write_JSONSchema` *Write a JSON Schema to file*

## Description

Serializes a schema built as a named list and writes it, with the keywords in the registry's reading order. Every generator that publishes to schema.rtemis.org writes through this, so the documents share one shape.

## Usage

```
write_JSONSchema(schema, file, overwrite = FALSE, digits = NA, verbosity = 1L)
```

## Arguments

| | |
|---|---|
| `schema` | Named list: The schema. |
| `file` | Character: Path to output JSON file. |
| `overwrite` | Logical: If TRUE, overwrite an existing file. |
| `digits` | Integer or NA: Significant digits for numeric values, passed to `jsonlite::toJSON()`. `NA` writes each number at full precision as R prints it. Pass `I(17)` where a value must round-trip an IEEE 754 double exactly: jsonlite's own default is 4 *decimal places*, which silently rounds, and a document that is nearly right is worse than one that is obviously wrong. |
| `verbosity` | Integer: Verbosity level. |

## Value

`schema`, invisibly.

## Author(s)

EDG

## Examples

```
schema <- list(
  `$schema` = "https://json-schema.org/draft/2020-12/schema",
  `$id` = "https://example.org/demo/v1/schema.json",
  title = "Demo",
  type = "object",
  properties = list(n = list(type = "integer", minimum = 1L))
)
tmpfile <- file.path(tempdir(), "demo.schema.json")
write_JSONSchema(schema, tmpfile, overwrite = TRUE, verbosity = 0L)
```

`write_lines` *Write lines to file*

## Description

Normalizes path, check if directory exists, creates it if necessary, writes lines to file, and checks if file was created successfully.

## Usage

```
write_lines(x, file, overwrite = FALSE, verbosity = 1L)
```

## Arguments

| | |
|---|---|
| `x` | Character: Text to write to file. |
| `file` | Character: Path to output file. |
| `overwrite` | Logical: If TRUE, overwrite an existing file. |
| `verbosity` | Integer: Verbosity level. |

## Value

Invisible NULL. Called for the side effect of writing to file.

## Author(s)

EDG

## Examples

```
tmpfile <- file.path(tempdir(), "demo.txt")
write_lines("hello", tmpfile, overwrite = TRUE, verbosity = 0L)
```

# Index


∗ **progress**
progress_begin, 93
progress_clear, 94
progress_end, 95
progress_lapply, 96
progress_update, 98

abbreviate_class, 5
abort, 6
abort(), *58*
ansi256_to_hex, 7
assert_config_contract, 8
assert_description_language, 9

base::debug(), *53*
bold, 10

character_scalar, 11
check_bounded_double_scalar, 12
check_bounded_integer_scalar, 13
check_character, 14
check_character_scalar, 15
check_data.table, 16
check_dependencies, 16
check_double_scalar, 17
check_enum, 18
check_file_exists, 18
check_float01exc, 19
check_float01inc, 20
check_float0pos, 20
check_float_neg1_1, 23
check_floatpos, 21
check_floatpos1, 22
check_inherits, 23
check_integer_scalar, *14*, 24
check_logical, 25
check_logical_scalar, 26
check_nonneg_double_scalar, 26
check_nonneg_double_vector, 27
check_numeric, 28
check_optional_bounded_double_scalar, 29
check_optional_bounded_integer_scalar, 30
check_optional_character_scalar, 31
check_optional_double_scalar, 31
check_optional_integer_scalar, 32
check_optional_logical_scalar, 33
check_optional_nonneg_double_scalar, 34
check_optional_nonneg_double_vector, 34
check_optional_pos_double_scalar, 35
check_optional_pos_double_vector, 36
check_optional_pos_integer_scalar, 37
check_optional_prob_scalar, 37
check_optional_prob_vector, 38
check_optional_scalar_character, 39
check_optional_unit_open_vector, 40
check_pos_double_scalar, *13*, 40
check_pos_double_vector, 41
check_pos_integer_scalar, 42
check_prob_scalar, *13*, 43
check_prob_vector, 43
check_scalar_character, 44
check_scalar_logical, 45
check_tabular, 46
check_unit_open_scalar, 46
check_unit_open_vector, 47
checkmark, 11
clean_colnames, 48
clean_int, 48
clean_names, 49
clean_posint, 50
col256, 51
collapse_head, 51
crossmark, 52

dbg, 53
ddSci, 54, *89*

double_scalar, 55

enum, 55

fmt, 56
fmt_gradient, 57
format_trace, 58
format_trace(), *7*

get_msg_sink, 59
get_msg_sink(), *109*, *113*
get_output_type, 60
get_output_type(), *10*, *12*, *51, 52*, *56, 57*, *62*, *93*, *97*, *107*
get_verbosity, 61

highlight, 61
html_div, 62
html_escape, 63
html_escape(), *65*
html_li, 63
html_p, 64
html_raw, 65
html_raw(), *67*
html_span, 65
html_strong, 66
html_tag, 67
html_ul, 68

info, 68
integer_scalar, 69

labelify, 70
lapply(), *96, 97*
logical_scalar, 71

match_arg, 71
msg, 72
msg0 *(*msg*)*, 72
msgdone, 73
msgstart, 74

nonneg_double_scalar, 75
nonneg_double_vector, 75
nonneg_integer_scalar, 76

optional, 77
optional_character_scalar, 77
optional_double_scalar, 78
optional_integer_scalar, 79
optional_logical_scalar, 79
optional_nonneg_double_scalar, 80
optional_nonneg_double_vector, 80
optional_nonneg_integer_scalar, 81
optional_pos_double_scalar, 82
optional_pos_double_vector, 82
optional_pos_integer_scalar, 83
optional_prob_scalar, 84
optional_prob_vector, 84
optional_unit_open_scalar, 85
optional_unit_open_vector, 86

plain, 86
pos_double_scalar, 87
pos_double_vector, 88
pos_integer_scalar, 88
printdf, 89
printls, 90
prob_scalar, 92
prob_vector, 92
progress_begin, 93
progress_begin(), *60*, *95–98*, *108*
progress_clear, 94
progress_clear(), *94*, *96–98*
progress_end, 95
progress_end(), *94, 95*, *97, 98*, *108*
progress_lapply, 96
progress_lapply(), *94–96*, *98*
progress_update, 98
progress_update(), *74*, *94–97*, *108*
prop_bag, 99
prop_boolean, 99
prop_const, 100
prop_float, *13*, 101
prop_integer, 102
prop_spec, 103
prop_spec(), *99–102*, *104*
prop_string, 104

repr, 105
repr_ls, 106
rtemis.core *(*rtemis.core-package*)*, 5
rtemis.core-package, 5
rtemis_colors, 107

set_msg_sink, 108
set_msg_sink(), *59*, *93*, *113*
strip_ansi, 109
success, 110

test_inherits, 110
traceback(), *58*

unit_open_scalar, 111
unit_open_vector, 112

warn, 112
with_msg_sink, 113
with_msg_sink(), *59*, *109*
write_JSONSchema, 114
write_lines, 115