

Package ‘stbl’

September 13, 2026

Title Stabilize Objects

Version 0.5.0

Description A set of consistent, opinionated functions to quickly check objects, coerce them to the desired configuration, or deliver informative error messages when that is not possible.

License MIT + file LICENSE

URL <https://stbl.wrangle.zone/>, <https://github.com/wranglezone/stbl>

BugReports <https://github.com/wranglezone/stbl/issues>

Depends R (>= 4.1)

Imports cli (>= 3.4.0), glue, methods, rlang (>= 1.0.3), vctrs

Suggests covr, hms, knitr, lubridate, rmarkdown, stringi, stringr, testthat (>= 3.3.0)

VignetteBuilder knitr

Config/Needs/website quarto

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

NeedsCompilation yes

Author Jon Harmon [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0003-4781-4346>>)

Maintainer Jon Harmon <jonthegeek@gmail.com>

Repository CRAN

Date/Publication 2026-09-13 16:20:02 UTC

Contents

are_chr_ish	4
are_dbl_ish	5
are_fct_ish	6
are_fn_ish	8
are_int_ish	9
are_lgl_ish	11
assert_contains	12
assert_not	14
assert_present	15
expect_pkg_error_classes	16
expect_pkg_error_snapshot	17
expect_pkg_message_classes	18
expect_pkg_message_snapshot	19
expect_pkg_warning_classes	20
expect_pkg_warning_snapshot	21
ignore_stbl_error	22
locale_datetime_formats	22
object_type	23
pkg_abort	24
pkg_inform	25
pkg_warn	26
regex_must_match	27
replace_stbl_error	28
specify_all_of	30
specify_any_of	31
specify_chr	31
specify_date	34
specify_dbl	35
specify_df	38
specify_dttm	40
specify_dur	42
specify_each	44
specify_fct	45
specify_int	47
specify_lgl	50
specify_lst	51
specify_one_of	54
specify_time	54
stabilize_all_of	56
stabilize_any_of	58
stabilize_arg	60
stabilize_chr	62
stabilize_chr_scalar	65
stabilize_date	68
stabilize_date_scalar	71
stabilize_dbl	73

stabilize_dbl_scalar	77
stabilize_df	81
stabilize_dttm	85
stabilize_dttm_scalar	88
stabilize_dur	91
stabilize_dur_scalar	95
stabilize_each	97
stabilize_fct	99
stabilize_fct_scalar	102
stabilize_int	104
stabilize_int_scalar	108
stabilize_lgl	112
stabilize_lgl_scalar	114
stabilize_lst	117
stabilize_one_of	121
stabilize_time	123
stabilize_time_scalar	126
to	128
to_chr	131
to_chr_scalar	133
to_date	135
to_date_scalar	137
to_dbl	139
to_dbl_scalar	141
to_df	143
to_dttm	144
to_dttm_scalar	147
to_dur	149
to_dur_scalar	151
to_each	153
to_fct	154
to_fct_scalar	156
to_fn	158
to_int	160
to_int_scalar	162
to_lgl	163
to_lgl_scalar	165
to_lst	167
to_time	168
to_time_scalar	171

are_chr_ish	<i>Check if an object can be safely coerced to character</i>
-------------	--

Description

`are_chr_ish()` is a vectorized predicate function that checks whether each element of its input can be safely coerced to a character vector. `are_character_ish()` is a synonym of `are_chr_ish()`.

`is_chr_ish()` is a scalar predicate function that checks if all elements of its input can be safely coerced to a character vector. `is_character_ish()` is a synonym of `is_chr_ish()`.

Usage

```
are_chr_ish(x, ...)
```

```
is_chr_ish(x, ...)
```

```
are_character_ish(x, ...)
```

```
is_character_ish(x, ...)
```

```
## Default S3 method:
```

```
are_chr_ish(x, ..., depth = 1)
```

Arguments

<code>x</code>	The object to check.
<code>...</code>	Arguments passed to methods.
<code>depth</code>	(integer(1)) Current recursion depth. Do not manually set this parameter.

Value

`are_chr_ish()` returns a logical vector with the same length as the input. `is_chr_ish()` returns a length-1 logical (TRUE or FALSE) for the entire vector.

See Also

Other character functions: [specify_chr\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [to\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#)

Other check functions: [are_dbl_ish\(\)](#), [are_fct_ish\(\)](#), [are_fn_ish\(\)](#), [are_int_ish\(\)](#), [are_lgl_ish\(\)](#)

Examples

```
are_chr_ish(letters)
```

```
is_chr_ish(letters)
```

```
are_chr_ish(1:10)
```

```
is_chr_ish(1:10)
```

```
are_chr_ish(list("a", 1, TRUE))
is_chr_ish(list("a", 1, TRUE))

are_chr_ish(list("a", 1, list(1, 2)))
is_chr_ish(list("a", 1, list(1, 2)))
```

are_dbl_ish	<i>Check if an object can be safely coerced to double</i>
-------------	---

Description

are_dbl_ish() is a vectorized predicate function that checks whether each element of its input can be safely coerced to a double vector. are_double_ish() is a synonym of are_dbl_ish().

is_dbl_ish() is a scalar predicate function that checks if all elements of its input can be safely coerced to a double vector. is_double_ish() is a synonym of is_dbl_ish().

Usage

```
are_dbl_ish(x, ...)

is_dbl_ish(x, ...)

are_double_ish(x, ...)

is_double_ish(x, ...)

## S3 method for class 'character'
are_dbl_ish(x, ..., coerce_character = TRUE)

## S3 method for class 'factor'
are_dbl_ish(x, ..., coerce_factor = TRUE)

## Default S3 method:
are_dbl_ish(x, ..., depth = 1)
```

Arguments

x	The object to check.
...	Arguments passed to methods.
coerce_character	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
coerce_factor	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while as.integer() and as.double() use the integer index of the factor.
depth	(integer(1)) Current recursion depth. Do not manually set this parameter.

Value

`are_dbl_ish()` returns a logical vector with the same length as the input. `is_dbl_ish()` returns a length-1 logical (TRUE or FALSE) for the entire vector.

See Also

Other double functions: `specify_dbl()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `to()`, `to_dbl()`, `to_dbl_scalar()`

Other check functions: `are_chr_ish()`, `are_fct_ish()`, `are_fn_ish()`, `are_int_ish()`, `are_lgl_ish()`

Examples

```
are_dbl_ish(c(1.0, 2.2, 3.14))
is_dbl_ish(c(1.0, 2.2, 3.14))
```

```
are_dbl_ish(1:3)
is_dbl_ish(1:3)
```

```
are_dbl_ish(c("1.1", "2.2", NA))
is_dbl_ish(c("1.1", "2.2", NA))
```

```
are_dbl_ish(c("a", "1.0"))
is_dbl_ish(c("a", "1.0"))
```

```
are_dbl_ish(list(1, "2.2", "c"))
is_dbl_ish(list(1, "2.2", "c"))
```

```
are_dbl_ish(c(1 + 1i, 1 + 0i, NA))
is_dbl_ish(c(1 + 1i, 1 + 0i, NA))
```

are_fct_ish

Check if an object can be safely coerced to a factor

Description

`are_fct_ish()` is a vectorized predicate function that checks whether each element of its input can be safely coerced to a factor. `are_factor_ish()` is a synonym of `are_fct_ish()`.

`is_fct_ish()` is a scalar predicate function that checks if all elements of its input can be safely coerced to a factor. `is_factor_ish()` is a synonym of `is_fct_ish()`.

Usage

```
are_fct_ish(x, ..., levels = NULL, to_na = character())
```

```
is_fct_ish(x, ..., levels = NULL, to_na = character(), max_levels = Inf)
```

```
are_factor_ish(x, ..., levels = NULL, to_na = character())
```

```
is_factor_ish(x, ..., levels = NULL, to_na = character(), max_levels = Inf)

## Default S3 method:
are_fct_ish(x, ..., levels = NULL, to_na = character(), depth = 1)
```

Arguments

x	The object to check.
...	Arguments passed to methods.
levels	(character) The desired factor levels. For factors, a vector's levels play the same role that <code>allowed_values</code> plays for other types: they restrict x to a fixed set of permitted values.
to_na	(character) Values to convert to NA.
max_levels	(numeric(1)) Maximum number of distinct non-NA values allowed across the whole vector after applying to_na.
depth	(integer(1)) Current recursion depth. Do not manually set this parameter.

Value

`are_fct_ish()` returns a logical vector with the same length as the input. `is_fct_ish()` returns a length-1 logical (TRUE or FALSE) for the entire vector.

See Also

Other factor functions: [specify_fct\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [to\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#)

Other check functions: [are_chr_ish\(\)](#), [are_dbl_ish\(\)](#), [are_fn_ish\(\)](#), [are_int_ish\(\)](#), [are_lgl_ish\(\)](#)

Examples

```
# When `levels` is `NULL`, atomic vectors are fct_ish, but nested lists are
# not.
are_fct_ish(c("a", 1, NA))
is_fct_ish(c("a", 1, NA))
are_fct_ish(list("a", list("b", "c")))
is_fct_ish(list("a", list("b", "c")))

# When `levels` is specified, values must be in `levels` or `to_na`.
are_fct_ish(c("a", "b", "c"), levels = c("a", "b"))
is_fct_ish(c("a", "b", "c"), levels = c("a", "b"))

# The `to_na` argument allows some values to be treated as `NA`.
are_fct_ish(c("a", "b", "z"), levels = c("a", "b"), to_na = "z")
is_fct_ish(c("a", "b", "z"), levels = c("a", "b"), to_na = "z")

# `max_levels` limits distinct non-`NA` values across the whole vector.
is_fct_ish(c("a", "b", "c"), max_levels = 2)

# Factors are also checked against the specified levels.
```

```
are_fct_ish(factor(c("a", "b", "c")), levels = c("a", "b"))
is_fct_ish(factor(c("a", "b", "c")), levels = c("a", "b"))
```

are_fn_ish

Check if an object can be safely coerced to a function

Description

are_fn_ish() is a vectorized predicate function that checks whether each element of its input can be safely coerced to a function. are_function_ish() is a synonym of are_fn_ish().

is_fn_ish() is a scalar predicate that checks whether its input can be safely coerced to a function by to_fn(). is_function_ish() is a synonym of is_fn_ish().

Usage

```
are_fn_ish(x, ...)

are_function_ish(x, ...)

is_fn_ish(x, ...)

is_function_ish(x, ...)
```

Arguments

x	The object to check.
...	Arguments passed to methods.

Details

are_fn_ish() returns TRUE for:

- Functions (including lambda functions created with ~ or \())
- Formulas (one-sided or two-sided, coercible via rlang::as_function())
- Character strings that are syntactically valid function names: either a bare name ("mean") or a namespaced name ("pkg::fn"). The check is syntactic; it does not verify that the named function exists in any environment.

is_fn_ish() returns TRUE for objects that to_fn() can coerce without error (assuming a matching function exists in the search path):

- Functions (including lambda functions created with ~ or \())
- One-sided or two-sided formulas
- Single non-NA character strings (including namespaced "pkg::fn" names)

NULL and length-0 character vectors are *not* considered function-ish because they do not represent a callable object; to_fn() converts them to NULL only as a permissive special case controlled by allow_null.

Value

are_fn_ish() returns a logical vector with the same length as the input. is_fn_ish() returns a length-1 logical (TRUE or FALSE).

See Also

Other function functions: [to\(\)](#), [to_fn\(\)](#)

Other check functions: [are_chr_ish\(\)](#), [are_dbl_ish\(\)](#), [are_fct_ish\(\)](#), [are_int_ish\(\)](#), [are_lgl_ish\(\)](#)

Examples

```
are_fn_ish(mean)
are_fn_ish(~ . + 1)
are_fn_ish(c("mean", "stats::median", NA, "", "1bad"))
are_fn_ish(NULL)
are_fn_ish(1L)

is_fn_ish(mean)
is_fn_ish("mean")
is_fn_ish("stats::median")
is_fn_ish(~ . + 1)
is_fn_ish(NULL)
is_fn_ish(1L)
```

are_int_ish

Check if an object can be safely coerced to integer

Description

are_int_ish() is a vectorized predicate function that checks whether each element of its input can be safely coerced to an integer vector. are_integer_ish() is a synonym of are_int_ish().

is_int_ish() is a scalar predicate function that checks if all elements of its input can be safely coerced to an integer vector. is_integer_ish() is a synonym of is_int_ish().

Usage

```
are_int_ish(x, ...)

is_int_ish(x, ...)

are_integer_ish(x, ...)

is_integer_ish(x, ...)

## S3 method for class 'character'
are_int_ish(x, ..., coerce_character = TRUE)
```

```
## S3 method for class 'factor'
are_int_ish(x, ..., coerce_factor = TRUE)
```

```
## Default S3 method:
are_int_ish(x, ..., depth = 1)
```

Arguments

<code>x</code>	The object to check.
<code>...</code>	Arguments passed to methods.
<code>coerce_character</code>	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while as.integer() and as.double() use the integer index of the factor.
<code>depth</code>	(integer(1)) Current recursion depth. Do not manually set this parameter.

Value

`are_int_ish()` returns a logical vector with the same length as the input. `is_int_ish()` returns a length-1 logical (TRUE or FALSE) for the entire vector.

See Also

Other integer functions: [specify_int\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [to\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#)

Other check functions: [are_chr_ish\(\)](#), [are_dbl_ish\(\)](#), [are_fct_ish\(\)](#), [are_fn_ish\(\)](#), [are_lgl_ish\(\)](#)

Examples

```
are_int_ish(1:4)
is_int_ish(1:4)

are_int_ish(c(1.0, 2.0, 3.00000))
is_int_ish(c(1.0, 2.0, 3.00000))

are_int_ish(c("1.0", "2.0", "3.00000"))
is_int_ish(c("1.0", "2.0", "3.00000"))

are_int_ish(c(1, 2.2, NA))
is_int_ish(c(1, 2.2, NA))

are_int_ish(c("1", "1.0", "1.1", "a"))
is_int_ish(c("1", "1.0", "1.1", "a"))

are_int_ish(factor(c("1", "a")))
is_int_ish(factor(c("1", "a")))
```

are_lgl_ish	<i>Check if an object can be safely coerced to logical</i>
-------------	--

Description

`are_lgl_ish()` is a vectorized predicate function that checks whether each element of its input can be safely coerced to a logical vector. `are_logical_ish()` is a synonym of `are_lgl_ish()`.

`is_lgl_ish()` is a scalar predicate function that checks if all elements of its input can be safely coerced to a logical vector. `is_logical_ish()` is a synonym of `is_lgl_ish()`.

Usage

```
are_lgl_ish(x, ...)

is_lgl_ish(x, ...)

are_logical_ish(x, ...)

is_logical_ish(x, ...)

## Default S3 method:
are_lgl_ish(x, ..., depth = 1)
```

Arguments

<code>x</code>	The object to check.
<code>...</code>	Arguments passed to methods.
<code>depth</code>	(integer(1)) Current recursion depth. Do not manually set this parameter.

Value

`are_lgl_ish()` returns a logical vector with the same length as the input. `is_lgl_ish()` returns a length-1 logical (TRUE or FALSE) for the entire vector.

See Also

Other logical functions: [specify_lgl\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [to\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#)

Other check functions: [are_chr_ish\(\)](#), [are_dbl_ish\(\)](#), [are_fct_ish\(\)](#), [are_fn_ish\(\)](#), [are_int_ish\(\)](#)

Examples

```
are_lgl_ish(c(TRUE, FALSE, NA))
is_lgl_ish(c(TRUE, FALSE, NA))

are_lgl_ish(c(1, 0, 1.0, NA))
is_lgl_ish(c(1, 0, 1.0, NA))
```

```

are_lgl_ish(c("T", "F", "TRUE", "FALSE", "true", "false", "1", "0"))
is_lgl_ish(c("T", "F", "TRUE", "FALSE", "true", "false", "1", "0"))

are_lgl_ish(c("T", "F", "a", "1.1"))
is_lgl_ish(c("T", "F", "a", "1.1"))

are_lgl_ish(factor(c("T", "a")))
is_lgl_ish(factor(c("T", "a")))

are_lgl_ish(list(TRUE, 0, "F", "a"))
is_lgl_ish(list(TRUE, 0, "F", "a"))

```

assert_contains	<i>Require that x contains a number of elements matching a specification</i>
-----------------	--

Description

assert_contains() applies spec to each element of x independently and counts how many elements match. It returns x unchanged if that count falls between min_matches and max_matches (inclusive), and errors otherwise.

Usage

```

assert_contains(
  x,
  spec,
  ...,
  min_matches = 1,
  max_matches = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

x	The object to stabilize.
spec	(function) A single stabilizer or coercion function, such as a to_* function (to_chr() , etc.), a stabilize_* function (stabilize_chr() , etc.), or a function produced by a specify_*() call (specify_chr() , etc.). Applied independently to each element of x.
...	Arguments passed to methods.
min_matches	(integer(1)) The minimum number of elements of x that must match spec. Must be >= 0. Set to 0 (with non-NULL max_matches) to check only an upper bound on the number of matches.

max_matches	(integer(1) or NULL) The maximum number of elements of x that may match spec. Must be >= min_matches. NULL (default) skips the upper-bound check.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

x, unchanged, if the number of elements of x matching spec is between min_matches and max_matches, or an error condition with classes <stbl-error>, <stbl-condition>, <rlang_error>, <error>, <condition>, and a specific class by failure mode:

- <stbl-error-too_few_matches> when fewer than min_matches elements match spec.
- <stbl-error-too_many_matches> when more than max_matches elements match spec.

See Also

Other stabilization functions: [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
# By default, at least 1 element must match spec
assert_contains(list("1", "a", "b"), stabilize_int)

# Require at least 2 matching elements
assert_contains(list("1", "2", "a"), stabilize_int, min_matches = 2)

# Errors because no elements are int-ish
try(assert_contains(list("a", "b"), stabilize_int))

# Errors because too many elements are int-ish
try(assert_contains(list("1", "2", "3"), stabilize_int, max_matches = 2))

# spec is applied to each element, not to x as a whole, so a scalar spec
# like stabilize_int_scalar() still matches every element of a list
assert_contains(
  list(1L, 2L, 3L),
  stabilize_int_scalar,
```

```

    min_matches = 2,
    max_matches = 3
  )

  # Use min_matches = 0 to check only an upper bound on the number of matches
  assert_contains(
    list("1", "a", "b"),
    stabilize_int,
    min_matches = 0,
    max_matches = 1
  )

```

 assert_not

Require a value not to match a specification

Description

assert_not() is the inverse of a single specification: it errors when x **would** be accepted by spec, and returns x unchanged otherwise.

Usage

```

assert_not(
  x,
  spec,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

x	The object to stabilize.
spec	A single stabilizer function, to_* function, specify_*() result, or assert_* function that x must not match.
...	Reserved for future use; must be empty.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

x, unchanged, if x does not match spec, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and `<stbl-error-matched_spec>` when x matches spec.

See Also

Other stabilization functions: `assert_contains()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
assert_not("a", specify_int())
assert_not(1.5, stabilize_int)

# Errors because "1" is int-ish, even though it isn't literally an integer
try(assert_not("1", specify_int()))

# Errors because stabilize_int() succeeds outright on 1L
try(assert_not(1L, stabilize_int))
```

assert_present	<i>Require a value to be non-NULL</i>
----------------	---------------------------------------

Description

`assert_present()` validates that a value is not NULL. Any non-NULL value passes through unchanged. This is useful as an element specification in `stabilize_lst()` when you need to require a named element without imposing any type constraints on its value.

Usage

```
assert_present(
  x,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The value, unchanged.

See Also

Other list functions: `specify_lst()`, `stabilize_lst()`, `to()`, `to_lst()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
assert_present("any value")
assert_present(list(1, 2, 3))
try(assert_present(NULL))

# Use as a named element spec in stabilize_lst()
stabilize_lst(list(data = mtcars), data = assert_present)
```

expect_pkg_error_classes

Test package error classes

Description

When you use `pkg_abort()` to signal errors, you can use this function to test that those errors are generated as expected.

Usage

```
expect_pkg_error_classes(object, package, ...)
```

Arguments

object	An expression that is expected to throw an error.
package	(character(1)) The name of the package to use in classes.
...	(character) Components of the class name, from least-specific to most.

Value

The classes of the error invisibly on success or the error on failure. Unlike most testthat expectations, this expectation cannot be usefully chained.

Examples

```
expect_pkg_error_classes(
  pkg_abort("stbl", "This is a test error", "test_subclass"),
  "stbl",
  "test_subclass"
)
try(
  expect_pkg_error_classes(
    pkg_abort("stbl", "This is a test error", "test_subclass"),
    "stbl",
    "different_subclass"
  )
)
```

```
expect_pkg_error_snapshot
```

Snapshot-test a package error

Description

A convenience wrapper around `testthat::expect_snapshot()` and `expect_pkg_error_classes()` that captures both the error class hierarchy and the user-facing message in a single snapshot.

Usage

```
expect_pkg_error_snapshot(
  object,
  package,
  ...,
  transform = NULL,
  variant = NULL,
  env = caller_env()
)
```

Arguments

object	An expression that is expected to throw an error.
package	(character(1)) The name of the package to use in classes.
...	(character) Components of the class name, from least-specific to most.
transform	(function or NULL) Optional function to scrub volatile output (e.g. temp paths) before snapshot comparison. Passed through to <code>testthat::expect_snapshot()</code> .
variant	(character(1) or NULL) Optional snapshot variant name. Passed through to <code>testthat::expect_snapshot()</code> .
env	(environment) The environment in which object should be evaluated. Assignments made inside object are visible to the caller after this function returns. <code>expect_pkg_error_classes()</code> is temporarily injected into env if it is not already findable, so this works even when this package is not attached.

Value

The result of `testthat::expect_snapshot()`, invisibly.

expect_pkg_message_classes

Test package message classes

Description

When you use `pkg_inform()` to signal messages, you can use this function to test that those messages are generated as expected.

Usage

```
expect_pkg_message_classes(object, package, ...)
```

Arguments

object	An expression that is expected to throw a message.
package	(character(1)) The name of the package to use in classes.
...	(character) Components of the class name, from least-specific to most.

Value

The message condition invisibly. Assignments made inside object (e.g. `result <- fn_that_informs()`) are visible to the caller after this function returns. Unlike most testthat expectations, this expectation cannot be usefully chained.

Examples

```
expect_pkg_message_classes(
  pkg_inform("stbl", "This is a test message", "test_subclass"),
  "stbl",
  "test_subclass"
)
try(
  expect_pkg_message_classes(
    pkg_inform("stbl", "This is a test message", "test_subclass"),
    "stbl",
    "different_subclass"
  )
)
```

```
expect_pkg_message_snapshot
```

Snapshot-test a package message

Description

A convenience wrapper around `testthat::expect_snapshot()` and `expect_pkg_message_classes()` that captures both the message class hierarchy and the user-facing message in a single snapshot.

Usage

```
expect_pkg_message_snapshot(
  object,
  package,
  ...,
  transform = NULL,
  variant = NULL,
  env = caller_env()
)
```

Arguments

<code>object</code>	An expression that is expected to throw a message.
<code>package</code>	(character(1)) The name of the package to use in classes.
<code>...</code>	(character) Components of the class name, from least-specific to most.
<code>transform</code>	(function or NULL) Optional function to scrub volatile output (e.g. temp paths) before snapshot comparison. Passed through to <code>testthat::expect_snapshot()</code> .
<code>variant</code>	(character(1) or NULL) Optional snapshot variant name. Passed through to <code>testthat::expect_snapshot()</code> .
<code>env</code>	(environment) The environment in which <code>object</code> should be evaluated. Assignments made inside <code>object</code> are visible to the caller after this function returns. <code>expect_pkg_message_classes()</code> is temporarily injected into <code>env</code> if it is not already findable, so this works even when this package is not attached.

Value

The result of `testthat::expect_snapshot()`, invisibly.

`expect_pkg_warning_classes`

Test package warning classes

Description

When you use `pkg_warn()` to signal warnings, you can use this function to test that those warnings are generated as expected.

Usage

```
expect_pkg_warning_classes(object, package, ...)
```

Arguments

<code>object</code>	An expression that is expected to throw a warning.
<code>package</code>	(character(1)) The name of the package to use in classes.
<code>...</code>	(character) Components of the class name, from least-specific to most.

Value

The warning condition invisibly. Assignments made inside `object` (e.g. `result <- fn_that_warns()`) are visible to the caller after this function returns. Unlike most `testthat` expectations, this expectation cannot be usefully chained.

Examples

```
expect_pkg_warning_classes(
  pkg_warn("stbl", "This is a test warning", "test_subclass"),
  "stbl",
  "test_subclass"
)
try(
  expect_pkg_warning_classes(
    pkg_warn("stbl", "This is a test warning", "test_subclass"),
    "stbl",
    "different_subclass"
  )
)
```

```
expect_pkg_warning_snapshot
  Snapshot-test a package warning
```

Description

A convenience wrapper around `testthat::expect_snapshot()` and `expect_pkg_warning_classes()` that captures both the warning class hierarchy and the user-facing message in a single snapshot.

Usage

```
expect_pkg_warning_snapshot(
  object,
  package,
  ...,
  transform = NULL,
  variant = NULL,
  env = caller_env()
)
```

Arguments

<code>object</code>	An expression that is expected to throw a warning.
<code>package</code>	(character(1)) The name of the package to use in classes.
<code>...</code>	(character) Components of the class name, from least-specific to most.
<code>transform</code>	(function or NULL) Optional function to scrub volatile output (e.g. temp paths) before snapshot comparison. Passed through to <code>testthat::expect_snapshot()</code> .
<code>variant</code>	(character(1) or NULL) Optional snapshot variant name. Passed through to <code>testthat::expect_snapshot()</code> .
<code>env</code>	(environment) The environment in which <code>object</code> should be evaluated. Assignments made inside <code>object</code> are visible to the caller after this function returns. <code>expect_pkg_warning_classes()</code> is temporarily injected into <code>env</code> if it is not already findable, so this works even when this package is not attached.

Value

The result of `testthat::expect_snapshot()`, invisibly.

ignore_stbl_error	<i>Ignore a stbl error class</i>
-------------------	----------------------------------

Description

Silences a stbl error with the specified subclass. Other errors (including other stbl errors) are not caught and propagate normally. See the documentation of each `stabilize_*`() or `to_*`() function for the types of errors it throws.

Usage

```
ignore_stbl_error(expr, subclass = character())
```

Arguments

expr	An expression to evaluate.
subclass	(character) The subclass(es) of the stbl error to ignore or replace. Combined with "stbl-error" to form the class name to intercept. For example, <code>c("coerce", "character")</code> catches errors of class <code>stbl-error-coerce-character</code> (stabilize_chr()), <code>c("coerce")</code> catches any <code>stbl-error-coerce</code> error, and <code>character()</code> (the default) catches any stbl error.

Value

The result of `expr` if no matching error is thrown, or `NULL` if a matching stbl error is caught.

Examples

```
ignore_stbl_error(to_chr(data.frame()), subclass = c("coerce", "character"))
ignore_stbl_error(to_chr("hello"), subclass = c("coerce", "character"))

# Omit subclass to catch any stbl error
ignore_stbl_error(to_chr(data.frame()))
```

locale_datetime_formats	<i>Determine locale-aware date and date-time formats</i>
-------------------------	--

Description

Returns an ordered set of `strptime()`-style format strings, used as the default value of `accepted_datetime_formats` in [stabilize_date\(\)](#), [stabilize_dttm\(\)](#), and their siblings. Character input is tried against each format in turn, in order, until one of them parses every element (see those functions for details).

Usage

```
locale_datetime_formats(locale_time = Sys.getlocale("LC_TIME"))
```

Arguments

locale_time (character(1)) LC_TIME locale, as returned by Sys.getlocale("LC_TIME"). Used to guess whether the locale's conventional date order is month-first (as in the United States) or day-first (most other locales); this is a coarse heuristic based on the locale string, not a full locale-aware calendar implementation.

Value

A character() vector of strptime()-style format strings, always starting with "%Y-%m-%dT%H:%M:%S", "%Y-%m-%d %H:%M:%S", and "%Y-%m-%d" (in that order), followed by the locale's conventional day/month order with "/" and then "-" separators (each listed with, then without, a " %H:%M:%S" suffix).

See Also

Other date functions: [specify_date\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#)
Other datetime functions: [specify_dttm\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#)

Examples

```
locale_datetime_formats("en_US.UTF-8")
locale_datetime_formats("en_GB.UTF-8")
```

object_type	<i>Identify the class, type, etc of an object</i>
-------------	---

Description

Extract the class (or type) of an object for use in error messages.

Usage

```
object_type(x)
```

Arguments

x An object to test.

Value

A length-1 character vector describing the class of the object.

Examples

```
object_type("a")
object_type(1L)
object_type(1.1)
object_type(mtcars)
object_type(rlang::quo(something))
```

pkg_abort

Signal an error with standards applied

Description

A wrapper around [cli::cli_abort\(\)](#) to throw classed errors, with an opinionated framework of error classes.

Usage

```
pkg_abort(
  package,
  message,
  subclass,
  call = caller_env(),
  message_env = call,
  parent = NULL,
  ...
)
```

Arguments

package	(character(1)) The name of the package to use in classes.
message	(character) The message for the new error. Messages will be formatted with cli::cli_bullets() .
subclass	(character) Class(es) to assign to the error. Will be prefixed by "{package}-error-".
call	(environment) The execution environment to mention as the source of error messages.
message_env	(environment) The execution environment to use to evaluate variables in error messages.
parent	A parent condition, as you might create during a rlang::try_fetch() . See rlang::abort() for additional information.
...	Additional parameters passed to cli::cli_abort() and on to rlang::abort() .

Value

NULL invisibly (called for side effects).

Examples

```
try(pkg_abort("stbl", "This is a test error", "test_subclass"))
tryCatch(
  pkg_abort("stbl", "This is a test error", "test_subclass"),
  `stbl-error` = function(e) {
    "Caught a generic stbl error."
  }
)
tryCatch(
  pkg_abort("stbl", "This is a test error", "test_subclass"),
  `stbl-error-test_subclass` = function(e) {
    "Caught a specific subclass of stbl error."
  }
)
```

pkg_inform

Signal a message with standards applied

Description

A wrapper around `cli::cli_inform()` to throw classed messages, with an opinionated framework of message classes.

Usage

```
pkg_inform(
  package,
  message,
  subclass,
  call = caller_env(),
  message_env = call,
  parent = NULL,
  ...
)
```

Arguments

package	(character(1)) The name of the package to use in classes.
message	(character) The message for the new message condition. Messages will be formatted with <code>cli::cli_bullets()</code> .
subclass	(character) Class(es) to assign to the message. Will be prefixed by "{package}-message-".
call	(environment) The execution environment to mention as the source of error messages.
message_env	(environment) The execution environment to use to evaluate variables in error messages.

parent A parent condition, as you might create during a `rlang::try_fetch()`. See `rlang::abort()` for additional information.

... Additional parameters passed to `cli::cli_inform()` and on to `rlang::inform()`.

Value

NULL invisibly (called for side effects).

Examples

```
pkg_inform("stbl", "This is a test message", "test_subclass")
withCallingHandlers(
  pkg_inform("stbl", "This is a test message", "test_subclass"),
  `stbl-message` = function(m) {
    message("Caught a generic stbl message.")
    invokeRestart("muffleMessage")
  }
)
withCallingHandlers(
  pkg_inform("stbl", "This is a test message", "test_subclass"),
  `stbl-message-test_subclass` = function(m) {
    message("Caught a specific subclass of stbl message.")
    invokeRestart("muffleMessage")
  }
)
```

pkg_warn

Signal a warning with standards applied

Description

A wrapper around `cli::cli_warn()` to throw classed warnings, with an opinionated framework of warning classes.

Usage

```
pkg_warn(
  package,
  message,
  subclass,
  call = caller_env(),
  message_env = call,
  parent = NULL,
  ...
)
```

Arguments

package	(character(1)) The name of the package to use in classes.
message	(character) The message for the new warning. Messages will be formatted with <code>cli::cli_bullets()</code> .
subclass	(character) Class(es) to assign to the warning. Will be prefixed by "{package}-warning-".
call	(environment) The execution environment to mention as the source of error messages.
message_env	(environment) The execution environment to use to evaluate variables in error messages.
parent	A parent condition, as you might create during a <code>rlang::try_fetch()</code> . See <code>rlang::abort()</code> for additional information.
...	Additional parameters passed to <code>cli::cli_warn()</code> and on to <code>rlang::warn()</code> .

Value

NULL invisibly (called for side effects).

Examples

```
pkg_warn("stbl", "This is a test warning", "test_subclass")
withCallingHandlers(
  pkg_warn("stbl", "This is a test warning", "test_subclass"),
  `stbl-warning` = function(w) {
    message("Caught a generic stbl warning.")
    invokeRestart("muffleWarning")
  }
)
withCallingHandlers(
  pkg_warn("stbl", "This is a test warning", "test_subclass"),
  `stbl-warning-test_subclass` = function(w) {
    message("Caught a specific subclass of stbl warning.")
    invokeRestart("muffleWarning")
  }
)
```

 regex_must_match

Create a regex matching rule

Description

Attach a standardized error message to a regex pattern. By default, the message will be "must match the regex pattern {regex}". If the input regex has a `negate` attribute set to `TRUE` (set automatically by `regex_must_not_match()`), the message will instead be "must not match...". This message can be used with `stabilize_chr()` and `stabilize_chr_scalar()`.

Usage

```
regex_must_match(regex)

regex_must_not_match(regex)
```

Arguments

regex (character) The regular expression pattern.

Value

For `regex_must_match`, the `regex` value with `names()` equal to the generated error message.

For `regex_must_not_match()`, the `regex` value with a `negate` attribute and with `names()` equal to the generated "must not match" error message.

Examples

```
regex_must_match("[aeiou]")

# With negation:
regex <- "[aeiou]"
attr(regex, "negate") <- TRUE
regex_must_match(regex)
regex_must_not_match("[aeiou]")
```

<code>replace_stbl_error</code>	<i>Replace a stbl error with a custom message</i>
---------------------------------	---

Description

Catches a stbl error with the specified subclass and replaces its message with a new one. Use this to provide more context-specific error messages when calling stbl functions inside your own functions. See the documentation of each `stabilize_*`() or `to_*`() function for the types of errors it throws.

Usage

```
replace_stbl_error(
  expr,
  message,
  subclass = character(),
  additional_class = character(),
  message_env = caller_env()
)
```

Arguments

expr	An expression to evaluate.
message	(character) The replacement error message. Formatted with <code>cli::cli_bullets()</code> .
subclass	(character) The subclass(es) of the stbl error to ignore or replace. Combined with "stbl-error" to form the class name to intercept. For example, <code>c("coerce", "character")</code> catches errors of class <code>stbl-error-coerce-character</code> (<code>stabilize_chr()</code>), <code>c("coerce")</code> catches any <code>stbl-error-coerce</code> error, and <code>character()</code> (the default) catches any stbl error.
additional_class	(character) Additional classes to prepend to the error class list. Useful for the class argument of <code>testthat::expect_error()</code> .
message_env	(environment) The execution environment to use to evaluate variables in error messages.

Value

The result of `expr`, or an error with the replacement message if a matching {stbl} error is thrown.

Examples

```
my_fn <- function(x) {
  x <- to_chr(x) |>
  replace_stbl_error(
    message = "{.arg x} must be a character vector of widgets",
    subclass = c("coerce", "character")
  )
}
try(my_fn(data.frame()))

# Specify a class to check for expected errors with testthat::expect_error()
my_fn2 <- function(x) {
  x <- to_chr(x) |>
  replace_stbl_error(
    message = "{.arg x} must be a character vector of widgets",
    subclass = c("coerce", "character"),
    additional_class = "mypkg-error-bad_widget"
  )
}
try(my_fn2(data.frame()))

# Omit subclass to catch any stbl error
my_fn3 <- function(x) {
  x <- to_chr(x) |>
  replace_stbl_error(message = "{.arg x} must be a character vector of widgets")
}
try(my_fn3(data.frame()))
```

specify_all_of	Create a specified all-of stabilizer function
----------------	---

Description

`specify_all_of()` creates a function that will call `stabilize_all_of()` with the provided specs. `specify_all_of()`'s function requires that `x` satisfy every provided spec, and that all specs agree on the coerced result.

Usage

```
specify_all_of(...)
```

Arguments

... Unnamed stabilizer functions, forwarded to `stabilize_all_of()`.

Value

A function of class "stbl_specified_fn" that calls `stabilize_all_of()` with the provided specs. The generated function will also accept ... for additional unnamed specs to pass to `stabilize_all_of()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other specification functions: `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_dbl()`, `specify_df()`, `specify_dttm()`, `specify_dur()`, `specify_each()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_a_then_b <- specify_all_of(
  specify_chr(regex = "^a"),
  specify_chr(regex = "b$")
)
stabilize_a_then_b("ab")
try(stabilize_a_then_b("ba"))
```

specify_any_of

*Create a specified any-of stabilizer function***Description**

specify_any_of() creates a function that will call [stabilize_any_of\(\)](#) with the provided specs. specify_any_of()'s function tries each spec in order and returns the result of the first one that succeeds.

Usage

```
specify_any_of(...)
```

Arguments

... Unnamed stabilizer / to_* / specify_*() specs, forwarded to [stabilize_any_of\(\)](#).

Value

A function of class "stbl_specified_fn" that calls [stabilize_any_of\(\)](#) with the provided specs. The generated function will also accept ... for additional unnamed specs to pass to [stabilize_any_of\(\)](#). You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other specification functions: [specify_all_of\(\)](#), [specify_chr\(\)](#), [specify_date\(\)](#), [specify_dbl\(\)](#), [specify_df\(\)](#), [specify_dttm\(\)](#), [specify_dur\(\)](#), [specify_each\(\)](#), [specify_fct\(\)](#), [specify_int\(\)](#), [specify_lgl\(\)](#), [specify_lst\(\)](#), [specify_one_of\(\)](#), [specify_time\(\)](#)

Examples

```
stabilize_int_or_chr <- specify_any_of(specify_int(), specify_chr())
stabilize_int_or_chr(1L)
stabilize_int_or_chr("a")
try(stabilize_int_or_chr(TRUE))
```

specify_chr

*Create a specified character stabilizer function***Description**

specify_chr() creates a function that will call [stabilize_chr\(\)](#) with the provided arguments. specify_chr_scalar() creates a function that will call [stabilize_chr_scalar\(\)](#) with the provided arguments. specify_character() is a synonym of specify_chr(), and specify_character_scalar() is a synonym of specify_chr_scalar().

Usage

```
specify_chr(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_characters = NULL,  
  max_characters = NULL,  
  regex = NULL,  
  allowed_values = NULL  
)  
  
specify_chr_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_characters = NULL,  
  max_characters = NULL,  
  regex = NULL,  
  allowed_values = NULL  
)  
  
specify_character(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_characters = NULL,  
  max_characters = NULL,  
  regex = NULL,  
  allowed_values = NULL  
)  
  
specify_character_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_characters = NULL,  
  max_characters = NULL,  
  regex = NULL,  
  allowed_values = NULL  
)
```

Arguments

allow_null (logical(1)) Is NULL an acceptable value?

allow_na	(logical(1)) Are NA values ok?
min_size	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
max_size	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
unique	(logical(1)) Should all elements in x be distinct?
min_characters	(integer(1)) Minimum number of characters allowed in each element.
max_characters	(integer(1)) Maximum number of characters allowed in each element.
regex	(character, list, or stringr_pattern) One or more optional regular expressions to test against the values of x. This can be a character vector, a list of character vectors, or a pattern object from the {stringr} package (e.g., <code>stringr::fixed("a.b")</code>). The default error message for non-matching values will include the pattern itself (see <code>regex_must_match()</code>). To provide a custom message, supply a named character vector where the value is the regex pattern and the name is the message that should be displayed. To check that a pattern is <i>not</i> matched, attach a negate attribute set to TRUE. If a complex regex pattern throws an error, try installing the stringi package.
allowed_values	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.
allow_zero_length	(logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls `stabilize_chr()` or `stabilize_chr_scalar()` with the provided arguments. The generated function will also accept ... for additional arguments to pass to `stabilize_chr()` or `stabilize_chr_scalar()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other character functions: `are_chr_ish()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `to()`, `to_chr()`, `to_chr_scalar()`

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_date()`, `specify_dbl()`, `specify_df()`, `specify_dttm()`, `specify_dur()`, `specify_each()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_email <- specify_chr(regex = "^[^@]+@[^@]+\\.([^@]+)$")
stabilize_email("stbl@example.com")
try(stabilize_email("not-an-email-address"))
```

specify_date

Create a specified date stabilizer function

Description

specify_date() creates a function that will call [stabilize_date\(\)](#) with the provided arguments. specify_date_scalar() creates a function that will call [stabilize_date_scalar\(\)](#) with the provided arguments.

Usage

```
specify_date(
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats()
)

specify_date_scalar(
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats()
)
```

Arguments

allow_null	(logical(1)) Is NULL an acceptable value?
allow_na	(logical(1)) Are NA values ok?
min_size	(integer(1)) The minimum size of the object. Object size will be tested using vctrs::vec_size() .
max_size	(integer(1)) The maximum size of the object. Object size will be tested using vctrs::vec_size() .
unique	(logical(1)) Should all elements in x be distinct?
min_value	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.

max_value (numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.

allowed_values A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.

accepted_datetime_formats (character) strptime()-style format strings to try, in order, when parsing a character x. The first format that parses every non-NA element of x is used; if none do, the result (and any error) is based on the first format tried. Defaults to [locale_datetime_formats\(\)](#), which starts with the unambiguous RFC 3339 shape ("%Y-%m-%d", optionally with a time-of-day component) before falling back to the current locale's conventional date order.

allow_zero_length (logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls [stabilize_date\(\)](#) or [stabilize_date_scalar\(\)](#) with the provided arguments. The generated function will also accept ... for additional arguments to pass to [stabilize_date\(\)](#) or [stabilize_date_scalar\(\)](#). You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other date functions: [locale_datetime_formats\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#)

Other specification functions: [specify_all_of\(\)](#), [specify_any_of\(\)](#), [specify_chr\(\)](#), [specify_dbl\(\)](#), [specify_df\(\)](#), [specify_dttm\(\)](#), [specify_dur\(\)](#), [specify_each\(\)](#), [specify_fct\(\)](#), [specify_int\(\)](#), [specify_lgl\(\)](#), [specify_lst\(\)](#), [specify_one_of\(\)](#), [specify_time\(\)](#)

Examples

```
stabilize_recent <- specify_date(min_value = "2000-01-01")
stabilize_recent("2024-01-01")
try(stabilize_recent("1999-12-31"))
```

specify_dbl

Create a specified double stabilizer function

Description

[specify_dbl\(\)](#) creates a function that will call [stabilize_dbl\(\)](#) with the provided arguments. [specify_dbl_scalar\(\)](#) creates a function that will call [stabilize_dbl_scalar\(\)](#) with the provided arguments. [specify_double\(\)](#) is a synonym of [specify_dbl\(\)](#), and [specify_double_scalar\(\)](#) is a synonym of [specify_dbl_scalar\(\)](#).

Usage

```
specify_dbl(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL  
)
```

```
specify_dbl_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL  
)
```

```
specify_double(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL  
)
```

```
specify_double_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL  
)
```

```

    allow_null = FALSE,
    allow_zero_length = FALSE,
    allow_na = TRUE,
    coerce_character = TRUE,
    coerce_factor = TRUE,
    min_value = NULL,
    max_value = NULL,
    exclusive_min_value = NULL,
    exclusive_max_value = NULL,
    allowed_values = NULL,
    multiple_of = NULL
  )

```

Arguments

<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>coerce_character</code>	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while <code>as.integer()</code> and <code>as.double()</code> use the integer index of the factor.
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>unique</code>	(logical(1)) Should all elements in x be distinct?
<code>min_value</code>	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.
<code>exclusive_min_value</code>	(numeric(1)) Similar to <code>max_value</code> , but x must be strictly greater than this value (>, not >=). NULL (default) values are not checked.
<code>exclusive_max_value</code>	(numeric(1)) Similar to <code>max_value</code> , but x must be strictly less than this value (<, not <=). NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>multiple_of</code>	(numeric(1), positive) x must be an integer multiple of this value. NULL (default) skips the check. For doubles, a small relative tolerance is applied to avoid floating-point false negatives.
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls `stabilize_dbl()` or `stabilize_dbl_scalar()` with the provided arguments. The generated function will also accept ... for additional arguments to pass to `stabilize_dbl()` or `stabilize_dbl_scalar()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other double functions: `are_dbl_ish()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `to()`, `to_dbl()`, `to_dbl_scalar()`

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_df()`, `specify_dttm()`, `specify_dur()`, `specify_each()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_3_to_5 <- specify_dbl(min_value = 3, max_value = 5)
stabilize_3_to_5(c(3.3, 4.4, 5))
try(stabilize_3_to_5(c(1:6)))
```

specify_df

Create a specified data frame stabilizer function

Description

`specify_df()` creates a function that will call `stabilize_df()` with the provided arguments. `specify_data_frame()` is a synonym of `specify_df()`.

Usage

```
specify_df(
  ...,
  .extra_cols = NULL,
  .col_names = NULL,
  .min_rows = NULL,
  .max_rows = NULL,
  .allow_null = TRUE,
  .allow_zero_length = TRUE,
  .required = ...names()
)
```

```
specify_data_frame(
  ...,
  .extra_cols = NULL,
  .col_names = NULL,
  .min_rows = NULL,
  .max_rows = NULL,
```

```

    .allow_null = TRUE,
    .allow_zero_length = TRUE,
    .required = ...names()
  )

```

Arguments

...	Named stabilizer functions, such as <code>stabilize_*</code> functions (<code>stabilize_chr()</code> , etc) or functions produced by <code>specify_*</code> () functions (<code>specify_chr()</code> , etc). Each name corresponds to a column in <code>.x</code> , and the function is used to validate that column when present. Whether the column is required is controlled by <code>.required</code> .
<code>.extra_cols</code>	Controls how columns of <code>.x</code> that are <i>not</i> explicitly listed in ... are handled. One of: • NULL or FALSE (default): any extra columns cause an error. • TRUE: extra columns are allowed, unchecked. • A single stabilizer function, such as a <code>stabilize_*</code> function (<code>stabilize_chr()</code>, etc) or a function produced by a <code>specify_*</code>() function (<code>specify_chr()</code>, etc), used to validate every extra column.
<code>.col_names</code>	(character) A character vector of column names that must be present in <code>.x</code> . Any columns listed here that are absent from <code>.x</code> will cause an error. Unlike ..., this does not validate the column contents.
<code>.min_rows</code>	(integer(1)) The minimum number of rows allowed in <code>.x</code> . If NULL (default), the row count is not checked.
<code>.max_rows</code>	(integer(1)) The maximum number of rows allowed in <code>.x</code> . If NULL (default), the row count is not checked.
<code>.allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>.allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>.required</code>	(character) Names (from ...) of columns that must be present in <code>.x</code> . Defaults to all names in ..., so every named spec is required unless you opt it out. Named specs <i>not</i> listed here are optional: if absent, no error is raised; if present, they're validated normally. Pass NULL or <code>character()</code> to make every named spec optional. A <code>.x</code> with zero columns skips this check when <code>.allow_zero_length = TRUE</code> (the default).

Value

A function of class "stbl_specified_fn" that calls `stabilize_df()` with the provided arguments. The generated function will also accept ... for additional named column specifications to pass to `stabilize_df()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other data frame functions: `stabilize_df()`, `to()`, `to_df()`

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_dbl()`, `specify_dttm()`, `specify_dur()`, `specify_each()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_person_df <- specify_df(
  name = specify_chr_scalar(allow_na = FALSE),
  age = specify_int_scalar(allow_na = FALSE),
  .extra_cols = assert_present
)
stabilize_person_df(data.frame(name = "Alice", age = 30L, score = 99.5))
try(stabilize_person_df(data.frame(name = "Alice")))
```

Mark a column as optional via .required

```
stabilize_person_df2 <- specify_df(
  name = specify_chr_scalar(allow_na = FALSE),
  age = specify_int_scalar(allow_na = FALSE),
  .required = "name"
)
stabilize_person_df2(data.frame(name = "Alice"))
```

specify_dttm

Create a specified datetime stabilizer function

Description

`specify_dttm()` creates a function that will call `stabilize_dttm()` with the provided arguments. `specify_dttm_scalar()` creates a function that will call `stabilize_dttm_scalar()` with the provided arguments. `specify_datetime()` is a synonym of `specify_dttm()`, and `specify_datetime_scalar()` is a synonym of `specify_dttm_scalar()`.

Usage

```
specify_dttm(
  tz = "UTC",
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats()
)

specify_dttm_scalar(
```

```

    tz = "UTC",
    allow_null = FALSE,
    allow_zero_length = FALSE,
    allow_na = TRUE,
    min_value = NULL,
    max_value = NULL,
    allowed_values = NULL,
    accepted_datetime_formats = locale_datetime_formats()
  )

```

```

specify_datetime(
  tz = "UTC",
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats()
)

```

```

specify_datetime_scalar(
  tz = "UTC",
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats()
)

```

Arguments

tz	(character(1)) The time zone to normalize x to. Must be "" or a value from OlsonNames() . Defaults to "UTC".
allow_null	(logical(1)) Is NULL an acceptable value?
allow_na	(logical(1)) Are NA values ok?
min_size	(integer(1)) The minimum size of the object. Object size will be tested using vctrs::vec_size() .
max_size	(integer(1)) The maximum size of the object. Object size will be tested using vctrs::vec_size() .
unique	(logical(1)) Should all elements in x be distinct?
min_value	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.

max_value (numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.

allowed_values A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.

accepted_datetime_formats (character) strptime()-style format strings to try, in order, when parsing a character x. The first format that parses every non-NA element of x is used; if none do, the result (and any error) is based on the first format tried. Defaults to [locale_datetime_formats\(\)](#), which starts with the unambiguous RFC 3339 shape ("%Y-%m-%d", optionally with a time-of-day component) before falling back to the current locale's conventional date order.

allow_zero_length (logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls [stabilize_dttm\(\)](#) or [stabilize_dttm_scalar\(\)](#) with the provided arguments. The generated function will also accept ... for additional arguments to pass to [stabilize_dttm\(\)](#) or [stabilize_dttm_scalar\(\)](#). You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other datetime functions: [locale_datetime_formats\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#)

Other specification functions: [specify_all_of\(\)](#), [specify_any_of\(\)](#), [specify_chr\(\)](#), [specify_date\(\)](#), [specify_dbl\(\)](#), [specify_df\(\)](#), [specify_dur\(\)](#), [specify_each\(\)](#), [specify_fct\(\)](#), [specify_int\(\)](#), [specify_lgl\(\)](#), [specify_lst\(\)](#), [specify_one_of\(\)](#), [specify_time\(\)](#)

Examples

```
stabilize_recent <- specify_dttm(min_value = "2000-01-01T00:00:00Z")
stabilize_recent("2024-01-01T00:00:00Z")
try(stabilize_recent("1999-12-31T00:00:00Z"))
```

specify_dur

Create a specified duration stabilizer function

Description

[specify_dur\(\)](#) creates a function that will call [stabilize_dur\(\)](#) with the provided arguments. [specify_dur_scalar\(\)](#) creates a function that will call [stabilize_dur_scalar\(\)](#) with the provided arguments. [specify_duration\(\)](#) is a synonym of [specify_dur\(\)](#), and [specify_duration_scalar\(\)](#) is a synonym of [specify_dur_scalar\(\)](#).

Usage

```
specify_dur(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL  
)  
  
specify_dur_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL  
)  
  
specify_duration(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL  
)  
  
specify_duration_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL  
)
```

Arguments

allow_null	(logical(1)) Is NULL an acceptable value?
allow_na	(logical(1)) Are NA values ok?
min_size	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .

max_size	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
unique	(logical(1)) Should all elements in x be distinct?
min_value	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.
max_value	(numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.
allowed_values	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.
allow_zero_length	(logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls `stabilize_dur()` or `stabilize_dur_scalar()` with the provided arguments. The generated function will also accept ... for additional arguments to pass to `stabilize_dur()` or `stabilize_dur_scalar()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other duration functions: `stabilize_dur()`, `stabilize_dur_scalar()`, `to_dur()`, `to_dur_scalar()`

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_dbl()`, `specify_df()`, `specify_dttm()`, `specify_each()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_short <- specify_dur(max_value = "P1D")
stabilize_short("PT12H")
try(stabilize_short("P2D"))
```

specify_each

Create a specified each stabilizer function

Description

`specify_each()` creates a function that will call `stabilize_each()` with the provided spec, letting an element-wise spec be reused and nested inside other `specify_*`/`stabilize_*` calls.

Usage

```
specify_each(spec)
```

Arguments

spec (function) A single stabilizer or coercion function, such as a `to_*` function (`to_chr()`, etc.), a `stabilize_*` function (`stabilize_chr()`, etc.), or a function produced by a `specify_*`() call (`specify_chr()`, etc.). Applied independently to each element of `x`.

Value

A function of class `"stbl_specified_fn"` that calls `stabilize_each()` with the provided `spec`. The generated function will also accept `simplify` and `...` to pass to `stabilize_each()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_dbl()`, `specify_df()`, `specify_dttm()`, `specify_dur()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_ints <- specify_each(stabilize_int)
stabilize_ints(list("1", "2", "3"))
try(stabilize_ints(list("1", "a")))
```

specify_fct

Create a specified factor stabilizer function

Description

`specify_fct()` creates a function that will call `stabilize_fct()` with the provided arguments. `specify_fct_scalar()` creates a function that will call `stabilize_fct_scalar()` with the provided arguments. `specify_factor()` is a synonym of `specify_fct()`, and `specify_factor_scalar()` is a synonym of `specify_fct_scalar()`.

Usage

```
specify_fct(
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  levels = NULL,
  to_na = character()
)

specify_fct_scalar(
```

```

    allow_null = FALSE,
    allow_zero_length = FALSE,
    allow_na = TRUE,
    levels = NULL,
    to_na = character()
  )

  specify_factor(
    allow_null = TRUE,
    allow_na = TRUE,
    min_size = NULL,
    max_size = NULL,
    levels = NULL,
    to_na = character()
  )

  specify_factor_scalar(
    allow_null = FALSE,
    allow_zero_length = FALSE,
    allow_na = TRUE,
    levels = NULL,
    to_na = character()
  )

```

Arguments

<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>levels</code>	(character) Expected levels. If NULL (default), the levels will be computed by <code>base::factor()</code> .
<code>to_na</code>	(character) Values to convert to NA.
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls `stabilize_fct()` or `stabilize_fct_scalar()` with the provided arguments. The generated function will also accept ... for additional arguments to pass to `stabilize_fct()` or `stabilize_fct_scalar()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other factor functions: `are_fct_ish()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `to()`, `to_fct()`, `to_fct_scalar()`

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_dbl()`, `specify_df()`, `specify_dttm()`, `specify_dur()`, `specify_each()`, `specify_int()`, `specify_lgl()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_lowercase_letter <- specify_fct(levels = letters)
stabilize_lowercase_letter(c("s", "t", "b", "l"))
try(stabilize_lowercase_letter("A"))
```

specify_int

*Create a specified integer stabilizer function***Description**

`specify_int()` creates a function that will call `stabilize_int()` with the provided arguments. `specify_int_scalar()` creates a function that will call `stabilize_int_scalar()` with the provided arguments. `specify_integer()` is a synonym of `specify_int()`, and `specify_integer_scalar()` is a synonym of `specify_int_scalar()`.

Usage

```
specify_int(
  allow_null = TRUE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL
)
```

```
specify_int_scalar(
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  coerce_character = TRUE,
```

```

    coerce_factor = TRUE,
    min_value = NULL,
    max_value = NULL,
    exclusive_min_value = NULL,
    exclusive_max_value = NULL,
    allowed_values = NULL,
    multiple_of = NULL
  )

```

```

specify_integer(
  allow_null = TRUE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL
)

```

```

specify_integer_scalar(
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL
)

```

Arguments

<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>coerce_character</code>	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor,

	while <code>as.integer()</code> and <code>as.double()</code> use the integer index of the factor.
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>unique</code>	(logical(1)) Should all elements in x be distinct?
<code>min_value</code>	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.
<code>exclusive_min_value</code>	(numeric(1)) Similar to <code>max_value</code> , but x must be strictly greater than this value (>, not >=). NULL (default) values are not checked.
<code>exclusive_max_value</code>	(numeric(1)) Similar to <code>max_value</code> , but x must be strictly less than this value (<, not <=). NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>multiple_of</code>	(numeric(1), positive) x must be an integer multiple of this value. NULL (default) skips the check. For doubles, a small relative tolerance is applied to avoid floating-point false negatives.
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls `stabilize_int()` or `stabilize_int_scalar()` with the provided arguments. The generated function will also accept ... for additional arguments to pass to `stabilize_int()` or `stabilize_int_scalar()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other integer functions: `are_int-ish()`, `stabilize_int()`, `stabilize_int_scalar()`, `to()`, `to_int()`, `to_int_scalar()`

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_dbl()`, `specify_df()`, `specify_dttm()`, `specify_dur()`, `specify_each()`, `specify_fct()`, `specify_lgl()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_3_to_5 <- specify_int(min_value = 3, max_value = 5)
stabilize_3_to_5(c(3:5))
try(stabilize_3_to_5(c(1:6)))
```

`specify_lgl`*Create a specified logical stabilizer function*

Description

`specify_lgl()` creates a function that will call `stabilize_lgl()` with the provided arguments. `specify_lgl_scalar()` creates a function that will call `stabilize_lgl_scalar()` with the provided arguments. `specify_logical()` is a synonym of `specify_lgl()`, and `specify_logical_scalar()` is a synonym of `specify_lgl_scalar()`.

Usage

```
specify_lgl(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  allowed_values = NULL  
)  
  
specify_lgl_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  allowed_values = NULL  
)  
  
specify_logical(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  allowed_values = NULL  
)  
  
specify_logical_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  allowed_values = NULL  
)
```

Arguments

<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?

min_size	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
max_size	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
allowed_values	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.
allow_zero_length	(logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls `stabilize_lgl()` or `stabilize_lgl_scalar()` with the provided arguments. The generated function will also accept ... for additional arguments to pass to `stabilize_lgl()` or `stabilize_lgl_scalar()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other logical functions: `are_lgl_ish()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `to()`, `to_lgl()`, `to_lgl_scalar()`

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_dbl()`, `specify_df()`, `specify_dttm()`, `specify_dur()`, `specify_each()`, `specify_fct()`, `specify_int()`, `specify_lst()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_few_lgl <- specify_lgl(max_size = 5)
stabilize_few_lgl(c(TRUE, "False", TRUE))
try(stabilize_few_lgl(rep(TRUE, 10)))
```

specify_lst

Create a specified list stabilizer function

Description

`specify_lst()` creates a function that will call `stabilize_lst()` with the provided arguments. `specify_list()` is a synonym of `specify_lst()`.

Usage

```
specify_lst(
  ...,
  .named = NULL,
  .unnamed = NULL,
  .unique = FALSE,
  .allow_null = TRUE,
```

```

    .allow_zero_length = TRUE,
    .min_size = NULL,
    .max_size = NULL,
    .required = ...names()
  )

specify_list(
  ...,
  .named = NULL,
  .unnamed = NULL,
  .unique = FALSE,
  .allow_null = TRUE,
  .allow_zero_length = TRUE,
  .min_size = NULL,
  .max_size = NULL,
  .required = ...names()
)

```

Arguments

...	Named stabilizer functions, such as <code>stabilize_*</code> functions (stabilize_chr() , etc) or functions produced by <code>specify_*</code> () functions (specify_chr() , etc). Each name corresponds to an element in <code>.x</code> , and the function is used to validate that element when present. Whether the element is required (its absence is an error) is controlled by <code>.required</code> .
<code>.named</code>	Controls how named elements of <code>.x</code> that are <i>not</i> explicitly listed in ... are handled. One of: • <code>NULL</code> or <code>FALSE</code> (default): any extra named elements cause an error. • <code>TRUE</code>: extra named elements are allowed, unchecked. • A single stabilizer function, such as a <code>stabilize_*</code> function (stabilize_chr(), etc) or a function produced by a <code>specify_*</code>() function (specify_chr(), etc), used to validate every extra named element.
<code>.unnamed</code>	Controls how unnamed elements of <code>.x</code> are handled. One of: • <code>NULL</code> or <code>FALSE</code> (default): any unnamed elements cause an error. • <code>TRUE</code>: unnamed elements are allowed, unchecked. • A single stabilizer function, such as a <code>stabilize_*</code> function (stabilize_chr(), etc) or a function produced by a <code>specify_*</code>() function (specify_chr(), etc), used to validate every unnamed element.
<code>.unique</code>	(logical(1)) Should all elements in <code>.x</code> be distinct? If <code>TRUE</code> , duplicated elements are rejected.
<code>.allow_null</code>	(logical(1)) Is <code>NULL</code> an acceptable value?
<code>.allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>.min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using vctrs::vec_size() .

<code>.max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>.required</code>	(character) Names (from ...) of elements that must be present in <code>.x</code> . Defaults to all names in ..., so every named spec is required unless you opt it out. Named specs <i>not</i> listed here are optional: if absent, no error is raised; if present, they're validated normally. Pass <code>NULL</code> or <code>character()</code> to make every named spec optional. A zero-length <code>.x</code> (such as <code>list()</code>) skips this check when <code>.allow_zero_length = TRUE</code> .

Value

A function of class "stbl_specified_fn" that calls `stabilize_lst()` with the provided arguments. The generated function will also accept ... for additional named element specifications to pass to `stabilize_lst()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other list functions: `assert_present()`, `stabilize_lst()`, `to()`, `to_lst()`

Other specification functions: `specify_all_of()`, `specify_any_of()`, `specify_chr()`, `specify_date()`, `specify_dbl()`, `specify_df()`, `specify_dttm()`, `specify_dur()`, `specify_each()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_one_of()`, `specify_time()`

Examples

```
stabilize_config <- specify_lst(
  name = specify_chr_scalar(allow_na = FALSE),
  version = stabilize_int_scalar,
  debug = specify_lgl_scalar(allow_na = FALSE),
  .unnamed = stabilize_chr_scalar
)
stabilize_config(list(name = "myapp", version = 1L, debug = FALSE, "extra"))
try(
  stabilize_config(
    list(name = "myapp", version = 1L, debug = FALSE, c("a", "b"))
  )
)

# Mark some elements as optional via .required
stabilize_settings <- specify_lst(
  name = specify_chr_scalar(),
  nickname = specify_chr_scalar(),
  .required = "name"
)
stabilize_settings(list(name = "Alice"))
```

specify_one_of	Create a specified one-of stabilizer function
----------------	---

Description

specify_one_of() creates a function that will call [stabilize_one_of\(\)](#) with the provided specs. specify_one_of()'s function requires that exactly one of the provided specs succeeds.

Usage

```
specify_one_of(...)
```

Arguments

... Unnamed stabilizer / to_* / specify_*() specs, forwarded to [stabilize_one_of\(\)](#).

Value

A function of class "stbl_specified_fn" that calls [stabilize_one_of\(\)](#) with the provided specs. The generated function will also accept ... for additional unnamed specs to pass to [stabilize_one_of\(\)](#). You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other specification functions: [specify_all_of\(\)](#), [specify_any_of\(\)](#), [specify_chr\(\)](#), [specify_date\(\)](#), [specify_dbl\(\)](#), [specify_df\(\)](#), [specify_dttm\(\)](#), [specify_dur\(\)](#), [specify_each\(\)](#), [specify_fct\(\)](#), [specify_int\(\)](#), [specify_lgl\(\)](#), [specify_lst\(\)](#), [specify_time\(\)](#)

Examples

```
stabilize_int_xor_chr <- specify_one_of(stabilize_int, stabilize_chr)
stabilize_int_xor_chr("a")
try(stabilize_int_xor_chr("1"))
```

specify_time	Create a specified time-of-day stabilizer function
--------------	--

Description

specify_time() creates a function that will call [stabilize_time\(\)](#) with the provided arguments. specify_time_scalar() creates a function that will call [stabilize_time_scalar\(\)](#) with the provided arguments.

Usage

```

specify_time(
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL
)

specify_time_scalar(
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL
)

```

Arguments

<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>unique</code>	(logical(1)) Should all elements in <code>x</code> be distinct?
<code>min_value</code>	(numeric(1)) The lowest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?

Value

A function of class "stbl_specified_fn" that calls `stabilize_time()` or `stabilize_time_scalar()` with the provided arguments. The generated function will also accept ... for additional arguments to pass to `stabilize_time()` or `stabilize_time_scalar()`. You can copy/paste the body of the resulting function if you want to provide additional context or functionality.

See Also

Other time functions: [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Other specification functions: [specify_all_of\(\)](#), [specify_any_of\(\)](#), [specify_chr\(\)](#), [specify_date\(\)](#), [specify_dbl\(\)](#), [specify_df\(\)](#), [specify_dttm\(\)](#), [specify_dur\(\)](#), [specify_each\(\)](#), [specify_fct\(\)](#), [specify_int\(\)](#), [specify_lgl\(\)](#), [specify_lst\(\)](#), [specify_one_of\(\)](#)

Examples

```
stabilize_afternoon <- specify_time(min_value = "12:00:00Z")
stabilize_afternoon("13:00:00Z")
try(stabilize_afternoon("06:00:00Z"))
```

stabilize_all_of	<i>Try to coerce or validate x as all of several specs</i>
------------------	--

Description

`stabilize_all_of()` validates and coerces `x` by applying every function in `...` to `x`, independently. `x` must satisfy every spec, and all specs must agree on the coerced result; if any spec fails, or specs disagree on the coerced value, an informative error is thrown. `stabilise_all_of()` is a synonym.

Usage

```
stabilize_all_of(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilise_all_of(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Unnamed stabilizer functions, such as <code>stabilize_*</code> functions (stabilize_chr() , etc.), <code>to_*</code> functions (to_chr() , etc.), functions produced by <code>specify_*</code> () calls (specify_chr() , etc.), or <code>assert_*</code> () functions (such as assert_not()) that return their input unchanged. Each is applied to the original <code>x</code> ; <code>x</code> must pass every one of them, and they must all return the same value.

<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

`x` coerced or validated by every function in ..., or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-empty_specs>` when no functions are supplied in ...
- `<stbl-error-named_spec>` when any element of ... is named.
- `<stbl-error-cant_stabilize_all_of>` when any provided function fails.
- `<stbl-error-inconsistent_all_of>` when every function succeeds, but they don't all produce the same coerced value.

See Also

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
# Returns x unchanged when all functions succeed
stabilize_all_of(1L, stabilize_int, stabilize_int_scalar)

# Each spec is applied to the original x, independently
stabilize_all_of("ab", specify_chr(regex = "^a"), specify_chr(regex = "b$"))

# Errors with a combined message when a spec fails
try(stabilize_all_of("a", stabilize_int, stabilize_chr))

# Errors because each spec sees the original "1" (a string); the second
# spec would succeed on 1L, but doesn't get the chance to see it
try(stabilize_all_of("1", stabilize_int, specify_dbl(coerce_character = FALSE)))

# Errors when specs succeed but disagree on the coerced value
# (stabilize_int() keeps 1L an integer; specify_dbl() makes it a double)
try(stabilize_all_of(1L, stabilize_int, specify_dbl()))
```

stabilize_any_of	<i>Try to coerce or validate x as one of several types</i>
------------------	--

Description

`stabilize_any_of()` attempts to validate and coerce `x` using each function in `...` in order. It returns the result of the first function that succeeds. If all functions fail, an informative error that combines the individual failure messages is thrown. `stabilise_any_of()` is a synonym.

`to_any_of()` is analogous to `to()`: it tries to coerce `x` to each type given in `...` (as a prototype such as `integer()` or `character()`) and returns the first successful result.

Usage

```
stabilize_any_of(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

```
stabilise_any_of(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

```
to_any_of(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	For <code>stabilize_any_of()</code> : unnamed stabilizer or coercion functions, such as <code>stabilize_*</code> functions (<code>stabilize_chr()</code> , etc.), <code>to_*</code> functions (<code>to_chr()</code> , etc.), functions produced by <code>specify_*</code> () calls (<code>specify_chr()</code> , etc.), or <code>assert_*</code> () functions (such as <code>assert_not()</code>) that return their input unchanged. For <code>to_any_of()</code> : prototype objects (e.g. <code>integer()</code> , <code>character()</code>) that determine the target types to try, passed as the <code>.to</code> argument of <code>to()</code> .

<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

`x` coerced or validated by the first successful function or prototype in ..., or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-empty_specs>` when no functions are supplied in ...
- `<stbl-error-named_spec>` when any element of ... is named.
- `<stbl-error-cant_stabilize_any_of>` when all provided functions fail.

See Also

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
# Returns x unchanged when the first function succeeds
stabilize_any_of(1L, stabilize_int, stabilize_chr)

# Falls through to stabilize_chr when stabilize_int fails
stabilize_any_of("a", stabilize_int, stabilize_chr)

# Coerces via the first matching function ("1" -> 1L)
stabilize_any_of("1", stabilize_int, stabilize_chr)

# A mixed list falls through to stabilize_chr because "a" can't be integer
stabilize_any_of(list(1L, "a"), stabilize_int, stabilize_chr)

# Errors with a combined message when all functions fail
try(stabilize_any_of(list(1, TRUE, "23", "maybe"), stabilize_lgl, stabilize_int))
# to_any_of() uses prototypes instead of functions
to_any_of(1L, integer(), character())
to_any_of("a", integer(), character())
to_any_of("1", integer(), character())
try(to_any_of(list(), integer(), character()))
```

stabilize_arg

Ensure an object meets expectations

Description

`stabilize_arg()` is used by other functions such as `stabilize_int()`. Use `stabilize_arg()` if the type-specific functions will not work for your use case, but you would still like to check things like size or whether the object is NULL.

`stabilize_arg_scalar()` is optimized to check for length-1 vectors.

Usage

```
stabilize_arg(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilize_arg_scalar(
  x,
  ...,
  allow_null = TRUE,
  allow_zero_length = TRUE,
  allow_na = TRUE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .

max_size	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
unique	(logical(1)) Should all elements in <code>x</code> be distinct?
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
allow_zero_length	(logical(1)) Are zero-length vectors acceptable?

Value

`x`, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE` in `stabilize_arg_scalar()`.
- `<stbl-error-non_scalar>` for non-scalar vectors in `stabilize_arg_scalar()`.

See Also

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
wrapper <- function(this_arg, ...) {
  stabilize_arg(this_arg, ...)
}
wrapper(1)
wrapper(NULL)
wrapper(NA)
try(wrapper(NULL, allow_null = FALSE))
try(wrapper(NA, allow_na = FALSE))
try(wrapper(1, min_size = 2))
```

```
try(wrapper(1:10, max_size = 5))
stabilize_arg_scalar("a")
stabilize_arg_scalar(1L)
try(stabilize_arg_scalar(1:10))
```

stabilize_chr	<i>Coerce to character with additional checks</i>
---------------	---

Description

Compared to `to_chr()`, `stabilize_chr()` checks more details, but is slower. `stabilise_chr()`, `stabilize_character()`, and `stabilise_character()` are synonyms of `stabilize_chr()`.

Usage

```
stabilize_chr(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_characters = NULL,
  max_characters = NULL,
  regex = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

```
stabilize_character(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_characters = NULL,
  max_characters = NULL,
  regex = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

```

)

stabilise_chr(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_characters = NULL,
  max_characters = NULL,
  regex = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilise_character(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_characters = NULL,
  max_characters = NULL,
  regex = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>unique</code>	(logical(1)) Should all elements in <code>x</code> be distinct?

<code>min_characters</code>	(integer(1)) Minimum number of characters allowed in each element.
<code>max_characters</code>	(integer(1)) Maximum number of characters allowed in each element.
<code>regex</code>	(character, list, or stringr_pattern) One or more optional regular expressions to test against the values of <code>x</code> . This can be a character vector, a list of character vectors, or a pattern object from the <code>{stringr}</code> package (e.g., <code>stringr::fixed("a.b")</code>). The default error message for non-matching values will include the pattern itself (see <code>regex_must_match()</code>). To provide a custom message, supply a named character vector where the value is the regex pattern and the name is the message that should be displayed. To check that a pattern is <i>not</i> matched, attach a <code>negate</code> attribute set to <code>TRUE</code> . If a complex regex pattern throws an error, try installing the <code>stringi</code> package.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). <code>NULL</code> (default) skips the check. <code>NA</code> values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a character vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-character>` when `x` cannot be coerced to character.
- `<stbl-error-bad_null>` for `NULL` values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for `NA` values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-duplicate_elements>` when `unique = TRUE` and duplicates are present.
- `<stbl-error-n_characters-too_few>` when elements have fewer characters than `min_characters`.
- `<stbl-error-n_characters-too_many>` when elements have more characters than `max_characters`.
- `<stbl-error-regex_mismatch>` when regex checks fail.
- `<stbl-error-allowed_values>` when values are not in `allowed_values`.

See Also

Other character functions: `are_chr-ish()`, `specify_chr()`, `stabilize_chr_scalar()`, `to()`, `to_chr()`, `to_chr_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`,

```

stabilize_dur(), stabilize_dur_scalar(), stabilize_fct(), stabilize_fct_scalar(), stabilize_int(),
stabilize_int_scalar(), stabilize_lgl(), stabilize_lgl_scalar(), stabilize_lst(), stabilize_one_of(),
stabilize_time(), stabilize_time_scalar(), to_chr(), to_chr_scalar(), to_date(), to_date_scalar(),
to_dbl(), to_dbl_scalar(), to_dttm(), to_dttm_scalar(), to_dur(), to_dur_scalar(), to_fct(),
to_fct_scalar(), to_int(), to_int_scalar(), to_lgl(), to_lgl_scalar(), to_time(), to_time_scalar()

```

Examples

```

stabilize_chr(letters)
stabilize_chr(1:10)
stabilize_chr(NULL)
try(stabilize_chr(NULL, allow_null = FALSE))
try(stabilize_chr(c("a", NA), allow_na = FALSE))
try(stabilize_chr(letters, min_size = 50))
try(stabilize_chr(letters, max_size = 20))
try(stabilize_chr(c("hi", "hey"), min_characters = 3))
try(stabilize_chr(c("hi", "hey"), max_characters = 2))
try(stabilize_chr(c("hide", "find", "find", "hide"), regex = "hide"))
try(stabilize_chr(c("a", "b", "z"), allowed_values = c("a", "b", "c")))

```

stabilize_chr_scalar *Coerce to length-1 character with additional checks*

Description

Checks whether a vector can be coerced to a length-1 character vector. `stabilize_chr_scalar()` is optimized to check for length-1 character vectors (compared to `stabilize_chr()` with `max_size = 1`). `stabilise_chr_scalar`, `stabilize_character_scalar()`, and `stabilise_character_scalar` are synonyms of `stabilize_chr_scalar()`.

Usage

```

stabilize_chr_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_characters = NULL,
  max_characters = NULL,
  regex = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilize_character_scalar(

```

```

x,
...,
allow_null = FALSE,
allow_zero_length = FALSE,
allow_na = TRUE,
min_characters = NULL,
max_characters = NULL,
regex = NULL,
allowed_values = NULL,
x_arg = caller_arg(x),
call = caller_env(),
x_class = object_type(x)
)

```

```

stabilise_chr_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_characters = NULL,
  max_characters = NULL,
  regex = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

```

stabilise_character_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_characters = NULL,
  max_characters = NULL,
  regex = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.

<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_characters</code>	(integer(1)) Minimum number of characters allowed in each element.
<code>max_characters</code>	(integer(1)) Maximum number of characters allowed in each element.
<code>regex</code>	(character, list, or stringr_pattern) One or more optional regular expressions to test against the values of <code>x</code> . This can be a character vector, a list of character vectors, or a pattern object from the <code>{stringr}</code> package (e.g., <code>stringr::fixed("a.b")</code>). The default error message for non-matching values will include the pattern itself (see regex_must_match()). To provide a custom message, supply a named character vector where the value is the regex pattern and the name is the message that should be displayed. To check that a pattern is <i>not</i> matched, attach a <code>negate</code> attribute set to <code>TRUE</code> . If a complex regex pattern throws an error, try installing the <code>stringi</code> package.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). <code>NULL</code> (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 character vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-character>` when `x` cannot be coerced to character.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-n_characters-too_few>` when the element has fewer characters than `min_characters`.
- `<stbl-error-n_characters-too_many>` when the element has more characters than `max_characters`.
- `<stbl-error-regex_mismatch>` when regex checks fail.
- `<stbl-error-allowed_values>` when the value is not in `allowed_values`.

See Also

Other character functions: [are_chr_ish\(\)](#), [specify_chr\(\)](#), [stabilize_chr\(\)](#), [to\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#)

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
stabilize_chr_scalar(TRUE)
try(stabilize_chr_scalar(c(TRUE, FALSE, TRUE)))
try(stabilize_chr_scalar(NULL))
stabilize_chr_scalar(NULL, allow_null = TRUE)
```

stabilize_date	<i>Coerce to date with additional checks</i>
----------------	--

Description

Compared to `to_date()`, `stabilize_date()` checks more details, but is slower. `stabilise_date()` is a synonym of `stabilize_date()`.

Usage

```
stabilize_date(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

```
stabilise_date(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
```

```

    min_size = NULL,
    max_size = NULL,
    unique = FALSE,
    min_value = NULL,
    max_value = NULL,
    allowed_values = NULL,
    accepted_datetime_formats = locale_datetime_formats(),
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>unique</code>	(logical(1)) Should all elements in <code>x</code> be distinct?
<code>min_value</code>	(numeric(1)) The lowest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>accepted_datetime_formats</code>	(character) <code>strptime()</code> -style format strings to try, in order, when parsing a character <code>x</code> . The first format that parses every non-NA element of <code>x</code> is used; if none do, the result (and any error) is based on the first format tried. Defaults to <code>locale_datetime_formats()</code> , which starts with the unambiguous RFC 3339 shape ("%Y-%m-%d", optionally with a time-of-day component) before falling back to the current locale's conventional date order.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a `base::Date` vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-date>` when `x` cannot be coerced to date.
- `<stbl-error-incompatible_values-date>` when some values cannot be safely converted to date.
- `<stbl-error-bad_null>` for `NULL` values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for `NA` values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-duplicate_elements>` when `unique = TRUE` and duplicates are present.
- `<stbl-error-outside_range>` when values fall outside `min_value` or `max_value`.
- `<stbl-error-allowed_values>` when values are not in `allowed_values`.

See Also

Other date functions: `locale_datetime_formats()`, `specify_date()`, `stabilize_date_scalar()`, `to_date()`, `to_date_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
stabilize_date(as.Date("2024-01-01"))
stabilize_date("2024-01-01")
stabilize_date(NULL)
try(stabilize_date(NULL, allow_null = FALSE))
try(stabilize_date(c(as.Date("2024-01-01"), NA), allow_na = FALSE))
stabilize_date(
  "11/13/2018",
  accepted_datetime_formats = locale_datetime_formats("en-US.UTF-8")
)
try(stabilize_date("2024-01-01", min_value = "2024-06-01"))
try(stabilize_date("2024-12-01", max_value = "2024-06-01"))
try(stabilize_date(
  c("2024-01-01", "2024-01-02"),
  allowed_values = "2024-01-01"
))
```

stabilize_date_scalar *Coerce to length-1 date with additional checks*

Description

Checks whether a vector can be coerced to a length-1 `base::Date` vector. `stabilize_date_scalar()` is optimized to check for length-1 date vectors (compared to `stabilize_date()` with `max_size = 1`). `stabilise_date_scalar` is a synonym of `stabilize_date_scalar()`.

Usage

```
stabilize_date_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL,  
  accepted_datetime_formats = locale_datetime_formats(),  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)  
  
stabilise_date_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL,  
  accepted_datetime_formats = locale_datetime_formats(),  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

allow_zero_length	(logical(1)) Are zero-length vectors acceptable?
allow_na	(logical(1)) Are NA values ok?
min_value	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.
max_value	(numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.
allowed_values	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.
accepted_datetime_formats	(character) strptime()-style format strings to try, in order, when parsing a character x. The first format that parses every non-NA element of x is used; if none do, the result (and any error) is based on the first format tried. Defaults to locale_datetime_formats() , which starts with the unambiguous RFC 3339 shape ("%Y-%m-%d", optionally with a time-of-day component) before falling back to the current locale's conventional date order.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 [base::Date](#) vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-date>` when x cannot be coerced to date.
- `<stbl-error-incompatible_values-date>` when some values cannot be safely converted to date.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-outside_range>` when values fall outside `min_value` or `max_value`.
- `<stbl-error-allowed_values>` when the value is not in `allowed_values`.

See Also

Other date functions: [locale_datetime_formats\(\)](#), [specify_date\(\)](#), [stabilize_date\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#)

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
stabilize_date_scalar(as.Date("2024-01-01"))
stabilize_date_scalar("2024-01-01")
try(stabilize_date_scalar(c("2024-01-01", "2024-01-02")))
try(stabilize_date_scalar(NULL))
stabilize_date_scalar(NULL, allow_null = TRUE)
```

stabilize_dbl

Coerce to double with additional checks

Description

Compared to `to_dbl()`, `stabilize_dbl()` checks more details, but is slower. `stabilise_dbl()`, `stabilize_double()`, and `stabilise_double()` are synonyms of `stabilize_dbl()`.

Usage

```
stabilize_dbl(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

```

stabilize_double(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

```

stabilise_dbl(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

```

stabilise_double(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  coerce_character = TRUE,

```

```

    coerce_factor = TRUE,
    min_size = NULL,
    max_size = NULL,
    unique = FALSE,
    min_value = NULL,
    max_value = NULL,
    exclusive_min_value = NULL,
    exclusive_max_value = NULL,
    allowed_values = NULL,
    multiple_of = NULL,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>coerce_character</code>	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while as.integer() and as.double() use the integer index of the factor.
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using vctrs::vec_size() .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using vctrs::vec_size() .
<code>unique</code>	(logical(1)) Should all elements in <code>x</code> be distinct?
<code>min_value</code>	(numeric(1)) The lowest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>exclusive_min_value</code>	(numeric(1)) Similar to <code>max_value</code> , but <code>x</code> must be strictly greater than this value (<code>></code> , not <code>>=</code>). NULL (default) values are not checked.
<code>exclusive_max_value</code>	(numeric(1)) Similar to <code>max_value</code> , but <code>x</code> must be strictly less than this value (<code><</code> , not <code><=</code>). NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .

<code>multiple_of</code>	(numeric(1), positive) <code>x</code> must be an integer multiple of this value. NULL (default) skips the check. For doubles, a small relative tolerance is applied to avoid floating-point false negatives.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a double vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-double>` when `x` cannot be coerced to double.
- `<stbl-error-incompatible_values-double>` when some values cannot be safely converted to double.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-duplicate_elements>` when `unique = TRUE` and duplicates are present.
- `<stbl-error-outside_range>` when values fall outside `min_value`, `max_value`, `exclusive_min_value`, or `exclusive_max_value`.
- `<stbl-error-allowed_values>` when values are not in `allowed_values`.
- `<stbl-error-not_multiple>` when values are not a multiple of `multiple_of`.

See Also

Other double functions: [are_dbl_ish\(\)](#), [specify_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [to\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```

stabilize_dbl(1:10)
stabilize_dbl("1.1")
stabilize_dbl(1 + 0i)
stabilize_dbl(NULL)
try(stabilize_dbl(NULL, allow_null = FALSE))
try(stabilize_dbl(c(1.1, NA), allow_na = FALSE))
try(stabilize_dbl(letters))
try(stabilize_dbl("1.1", coerce_character = FALSE))
try(stabilize_dbl(factor(c("1.1", "a"))))
try(stabilize_dbl(factor("1.1"), coerce_factor = FALSE))
try(stabilize_dbl(1:10, min_value = 3.5))
try(stabilize_dbl(1:10, max_value = 7.5))
try(stabilize_dbl(1:10, exclusive_min_value = 3))
try(stabilize_dbl(1:10, exclusive_max_value = 8))
try(stabilize_dbl(c(1.1, 2.2, 3.3), allowed_values = c(1.1, 2.2)))
try(stabilize_dbl(c(0.1, 0.25), multiple_of = 0.05))

```

stabilize_dbl_scalar *Coerce to length-1 double with additional checks*

Description

Checks whether a vector can be coerced to a length-1 double vector. `stabilize_dbl_scalar()` is optimized to check for length-1 double vectors (compared to `stabilize_dbl()` with `max_size = 1`). `stabilise_dbl_scalar`, `stabilize_double_scalar()`, and `stabilise_double_scalar` are synonyms of `stabilize_dbl_scalar()`.

Usage

```

stabilize_dbl_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

```
stabilize_double_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_dbl_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_double_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_value = NULL,
```

```

    max_value = NULL,
    exclusive_min_value = NULL,
    exclusive_max_value = NULL,
    allowed_values = NULL,
    multiple_of = NULL,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>coerce_character</code>	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while as.integer() and as.double() use the integer index of the factor.
<code>min_value</code>	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.
<code>exclusive_min_value</code>	(numeric(1)) Similar to <code>max_value</code> , but x must be strictly greater than this value (>, not >=). NULL (default) values are not checked.
<code>exclusive_max_value</code>	(numeric(1)) Similar to <code>max_value</code> , but x must be strictly less than this value (<, not <=). NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>multiple_of</code>	(numeric(1), positive) x must be an integer multiple of this value. NULL (default) skips the check. For doubles, a small relative tolerance is applied to avoid floating-point false negatives.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.

`x_class` (character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 double vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-double>` when `x` cannot be coerced to double.
- `<stbl-error-incompatible_values-double>` when some values cannot be safely converted to double.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-outside_range>` when values fall outside `min_value`, `max_value`, `exclusive_min_value`, or `exclusive_max_value`.
- `<stbl-error-allowed_values>` when the value is not in `allowed_values`.
- `<stbl-error-not_multiple>` when the value is not a multiple of `multiple_of`.

See Also

Other double functions: [are_dbl-ish\(\)](#), [specify_dbl\(\)](#), [stabilize_dbl\(\)](#), [to\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_dbl_scalar(1.0)
stabilize_dbl_scalar("1.1")
try(stabilize_dbl_scalar(1:10))
try(stabilize_dbl_scalar(NULL))
stabilize_dbl_scalar(NULL, allow_null = TRUE)
```

stabilize_df	<i>Ensure a data frame meets expectations</i>
--------------	---

Description

`stabilize_df()` validates the structure and contents of a data frame. It can check that specific named columns are present and valid, that extra columns conform to a shared rule, that required column names are present, and that the row count is within specified bounds. `stabilise_df()`, `stabilize_data_frame()`, and `stabilise_data_frame()` are synonyms of `stabilize_df()`.

Usage

```
stabilize_df(  
  .x,  
  ...,  
  .extra_cols = NULL,  
  .col_names = NULL,  
  .min_rows = NULL,  
  .max_rows = NULL,  
  .allow_null = TRUE,  
  .allow_zero_length = TRUE,  
  .required = ...names(),  
  .x_arg = caller_arg(.x),  
  .call = caller_env(),  
  .x_class = object_type(.x)  
)
```

```
stabilise_df(  
  .x,  
  ...,  
  .extra_cols = NULL,  
  .col_names = NULL,  
  .min_rows = NULL,  
  .max_rows = NULL,  
  .allow_null = TRUE,  
  .allow_zero_length = TRUE,  
  .required = ...names(),  
  .x_arg = caller_arg(.x),  
  .call = caller_env(),  
  .x_class = object_type(.x)  
)
```

```
stabilize_data_frame(  
  .x,  
  ...,  
  .extra_cols = NULL,  
  .col_names = NULL,
```

```

    .min_rows = NULL,
    .max_rows = NULL,
    .allow_null = TRUE,
    .allow_zero_length = TRUE,
    .required = ...names(),
    .x_arg = caller_arg(.x),
    .call = caller_env(),
    .x_class = object_type(.x)
)

```

```

stabilise_data_frame(
  .x,
  ...,
  .extra_cols = NULL,
  .col_names = NULL,
  .min_rows = NULL,
  .max_rows = NULL,
  .allow_null = TRUE,
  .allow_zero_length = TRUE,
  .required = ...names(),
  .x_arg = caller_arg(.x),
  .call = caller_env(),
  .x_class = object_type(.x)
)

```

Arguments

<code>.x</code>	The object to stabilize.
<code>...</code>	Named stabilizer functions, such as <code>stabilize_*</code> functions (stabilize_chr() , etc) or functions produced by <code>specify_*</code> () functions (specify_chr() , etc). Each name corresponds to a column in <code>.x</code> , and the function is used to validate that column when present. Whether the column is required is controlled by <code>.required</code> .
<code>.extra_cols</code>	Controls how columns of <code>.x</code> that are <i>not</i> explicitly listed in <code>...</code> are handled. One of: • <code>NULL</code> or <code>FALSE</code> (default): any extra columns cause an error. • <code>TRUE</code>: extra columns are allowed, unchecked. • A single stabilizer function, such as a <code>stabilize_*</code> function (stabilize_chr(), etc) or a function produced by a <code>specify_*</code>() function (specify_chr(), etc), used to validate every extra column.
<code>.col_names</code>	(character) A character vector of column names that must be present in <code>.x</code> . Any columns listed here that are absent from <code>.x</code> will cause an error. Unlike <code>...</code> , this does not validate the column contents.
<code>.min_rows</code>	(integer(1)) The minimum number of rows allowed in <code>.x</code> . If <code>NULL</code> (default), the row count is not checked.
<code>.max_rows</code>	(integer(1)) The maximum number of rows allowed in <code>.x</code> . If <code>NULL</code> (default), the row count is not checked.

<code>.allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>.allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>.required</code>	(character) Names (from ...) of columns that must be present in <code>.x</code> . Defaults to all names in ..., so every named spec is required unless you opt it out. Named specs <i>not</i> listed here are optional: if absent, no error is raised; if present, they're validated normally. Pass NULL or <code>character()</code> to make every named spec optional. A <code>.x</code> with zero columns skips this check when <code>.allow_zero_length = TRUE</code> (the default).
<code>.x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>.call</code>	(environment) The execution environment to mention as the source of error messages.
<code>.x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The validated data frame, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-bad_null>` for NULL values when `.allow_null = FALSE`.
- `<stbl-error-coerce-data.frame>` when `.x` cannot be coerced to a data frame.
- `<stbl-error-jagged>` when list input has incompatible element lengths for conversion to a data frame.
- `<stbl-error-too_few_rows>` when the data frame has fewer than `.min_rows` rows.
- `<stbl-error-too_many_rows>` when the data frame has more than `.max_rows` rows.
- `<stbl-error-missing_cols>` when required names in `.col_names` are absent.
- `<stbl-error-unnamed_spec>` when any element passed through ... is unnamed.
- `<stbl-error-missing_element>`, `<stbl-error-bad_named>`, `<stbl-error-bad_unnamed>`, or `<stbl-error-duplicate_names>` when list-like column checks delegated to `stabilize_lst()` fail.

See Also

Other data frame functions: `specify_df()`, `to()`, `to_df()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```

# Basic validation: required columns with type specs
stabilize_df(
  data.frame(name = "Alice", age = 30L),
  name = specify_chr_scalar(),
  age = specify_int_scalar()
)

# Validate extra columns with .extra_cols
stabilize_df(
  data.frame(name = "Alice", age = 30L, score = 99.5),
  name = specify_chr_scalar(),
  age = specify_int_scalar(),
  .extra_cols = assert_present
)

# Allow extra columns unchecked with .extra_cols = TRUE
stabilize_df(
  data.frame(name = "Alice", age = 30L, score = 99.5),
  name = specify_chr_scalar(),
  .extra_cols = TRUE
)

# Check required column names without validating contents
stabilize_df(
  mtcars,
  .col_names = c("mpg", "cyl"),
  .extra_cols = assert_present
)

# Enforce row count constraints
try(
  stabilize_df(mtcars[0, ], .min_rows = 1, .extra_cols = assert_present)
)

# NULL is allowed by default
stabilize_df(NULL)
try(stabilize_df(NULL, .allow_null = FALSE))

# Coercible inputs such as named lists are accepted
stabilize_df(
  list(name = "Alice", age = 30L),
  name = specify_chr_scalar(),
  age = specify_int_scalar()
)

# Non-coercible inputs are rejected
try(stabilize_df("not a data frame"))

# Mark a column as optional via .required
stabilize_df(
  data.frame(name = "Alice"),

```

```

    name = specify_chr_scalar(),
    age = specify_int_scalar(),
    .required = "name"
  )

```

stabilize_dttm

Coerce to date-time with additional checks

Description

Compared to [to_dttm\(\)](#), `stabilize_dttm()` checks more details, but is slower. `stabilise_dttm()`, `stabilize_datetime()`, `stabilise_dttm()`, `stabilize_datetime()`, and `stabilise_datetime()` are synonyms of `stabilize_dttm()`.

Usage

```

stabilize_dttm(
  x,
  ...,
  tz = "UTC",
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

```

stabilise_dttm(
  x,
  ...,
  tz = "UTC",
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats(),

```

```

    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

stabilize_datetime(
  x,
  ...,
  tz = "UTC",
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilise_datetime(
  x,
  ...,
  tz = "UTC",
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>tz</code>	(character(1)) The time zone to normalize <code>x</code> to. Must be "" or a value from OlsonNames() . Defaults to "UTC".
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

allow_na	(logical(1)) Are NA values ok?
min_size	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
max_size	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
unique	(logical(1)) Should all elements in x be distinct?
min_value	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.
max_value	(numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.
allowed_values	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.
accepted_datetime_formats	(character) <code>strptime()</code> -style format strings to try, in order, when parsing a character x. The first format that parses every non-NA element of x is used; if none do, the result (and any error) is based on the first format tried. Defaults to <code>locale_datetime_formats()</code> , which starts with the unambiguous RFC 3339 shape ("%Y-%m-%d", optionally with a time-of-day component) before falling back to the current locale's conventional date order.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a `base::POSIXct` vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-datetime>` when x cannot be coerced to datetime.
- `<stbl-error-incompatible_values-datetime>` when some values cannot be safely converted to datetime.
- `<stbl-error-bad_tz>` when tz is not "" or a value from `OlsonNames()`.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-duplicate_elements>` when `unique = TRUE` and duplicates are present.
- `<stbl-error-outside_range>` when values fall outside `min_value` or `max_value`.
- `<stbl-error-allowed_values>` when values are not in `allowed_values`.

See Also

Other datetime functions: [locale_datetime_formats\(\)](#), [specify_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_dttm(as.POSIXct("2024-01-01 12:00:00", tz = "UTC"))
stabilize_dttm("2024-01-01T12:00:00Z")
stabilize_dttm(NULL)
try(stabilize_dttm(NULL, allow_null = FALSE))
try(stabilize_dttm(
  c("2024-01-01T12:00:00Z", NA),
  allow_na = FALSE
))
stabilize_dttm("2024-01-01 12:00:00")
try(stabilize_dttm(
  "2024-01-01T12:00:00Z",
  min_value = "2024-06-01T00:00:00Z"
))
try(stabilize_dttm(
  "2024-12-01T00:00:00Z",
  max_value = "2024-06-01T00:00:00Z"
))
```

`stabilize_dttm_scalar` *Coerce to length-1 date-time with additional checks*

Description

Checks whether a vector can be coerced to a length-1 [base::POSIXct](#) vector. `stabilize_dttm_scalar()` is optimized to check for length-1 date-time vectors (compared to [stabilize_dttm\(\)](#) with `max_size = 1`). `stabilise_dttm_scalar` are synonyms of `stabilize_dttm_scalar()`.

Usage

```
stabilize_dttm_scalar(
  x,
  ...,
  tz = "UTC",
  allow_null = FALSE,
```

```

    allow_zero_length = FALSE,
    allow_na = TRUE,
    min_value = NULL,
    max_value = NULL,
    allowed_values = NULL,
    accepted_datetime_formats = locale_datetime_formats(),
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

```

```

stabilise_dttm_scalar(
  x,
  ...,
  tz = "UTC",
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

```

stabilize_datetime_scalar(
  x,
  ...,
  tz = "UTC",
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  accepted_datetime_formats = locale_datetime_formats(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

```

stabilise_datetime_scalar(
  x,
  ...,
  tz = "UTC",
  allow_null = FALSE,

```

```

    allow_zero_length = FALSE,
    allow_na = TRUE,
    min_value = NULL,
    max_value = NULL,
    allowed_values = NULL,
    accepted_datetime_formats = locale_datetime_formats(),
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
  )

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>tz</code>	(character(1)) The time zone to normalize <code>x</code> to. Must be "" or a value from OlsonNames() . Defaults to "UTC".
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_value</code>	(numeric(1)) The lowest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>accepted_datetime_formats</code>	(character) <code>strptime()</code> -style format strings to try, in order, when parsing a character <code>x</code> . The first format that parses every non-NA element of <code>x</code> is used; if none do, the result (and any error) is based on the first format tried. Defaults to locale_datetime_formats() , which starts with the unambiguous RFC 3339 shape ("%Y-%m-%d", optionally with a time-of-day component) before falling back to the current locale's conventional date order.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 [base::POSIXct](#) vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-datetime>` when `x` cannot be coerced to datetime.
- `<stbl-error-incompatible_values-datetime>` when some values cannot be safely converted to datetime.
- `<stbl-error-bad_tz>` when `tz` is not `""` or a value from [OlsonNames\(\)](#).
- `<stbl-error-bad_null>` for `NULL` values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for `NA` values when `allow_na = FALSE`.
- `<stbl-error-outside_range>` when values fall outside `min_value` or `max_value`.
- `<stbl-error-allowed_values>` when the value is not in `allowed_values`.

See Also

Other datetime functions: [locale_datetime_formats\(\)](#), [specify_dttm\(\)](#), [stabilize_dttm\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_dttm_scalar(as.POSIXct("2024-01-01 12:00:00", tz = "UTC"))
stabilize_dttm_scalar("2024-01-01T12:00:00Z")
try(stabilize_dttm_scalar(c(
  "2024-01-01T12:00:00Z",
  "2024-01-02T12:00:00Z"
)))
try(stabilize_dttm_scalar(NULL))
stabilize_dttm_scalar(NULL, allow_null = TRUE)
```

stabilize_dur

Coerce to duration with additional checks

Description

Compared to [to_dur\(\)](#), [stabilize_dur\(\)](#) checks more details, but is slower. [stabilise_dur\(\)](#), [stabilize_duration\(\)](#), and [stabilise_duration\(\)](#) are synonyms of [stabilize_dur\(\)](#).

Usage

```
stabilize_dur(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_dur(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilize_duration(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```

stabilise_duration(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>unique</code>	(logical(1)) Should all elements in <code>x</code> be distinct?
<code>min_value</code>	(numeric(1)) The lowest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Details

`min_value`, `max_value`, and `allowed_values` comparisons rely on lubridate's approximate, nominal lengths for years and months (see [to_dur\(\)](#)); treat range checks on calendar-aware durations as approximate.

Value

The input as a [lubridate::Period](#) vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-duration>` when `x` cannot be coerced to a duration.
- `<stbl-error-incompatible_values-duration>` when some values cannot be safely converted to a duration.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-duplicate_elements>` when `unique = TRUE` and duplicates are present.
- `<stbl-error-outside_range>` when values fall outside `min_value` or `max_value`.
- `<stbl-error-allowed_values>` when values are not in `allowed_values`.

See Also

Other duration functions: [specify_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_dur(lubridate::period(day = 1))
stabilize_dur("P3Y6M4DT12H30M5S")
stabilize_dur(NULL)
try(stabilize_dur(NULL, allow_null = FALSE))
try(stabilize_dur(c("P1D", NA), allow_na = FALSE))
try(stabilize_dur("P"))
try(stabilize_dur("P1D", min_value = "P2D"))
try(stabilize_dur("P2D", max_value = "P1D"))
```

stabilize_dur_scalar	<i>Coerce to length-1 duration with additional checks</i>
----------------------	---

Description

Checks whether a vector can be coerced to a length-1 [lubridate::Period](#) vector. `stabilize_dur_scalar()` is optimized to check for length-1 duration vectors (compared to `stabilize_dur()` with `max_size = 1`). `stabilise_dur_scalar`, `stabilize_duration_scalar`, and `stabilise_duration_scalar` are synonyms of `stabilize_dur_scalar()`.

Usage

```
stabilize_dur_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_dur_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  allowed_values = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilize_duration_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  min_value = NULL,
```

```

    max_value = NULL,
    allowed_values = NULL,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

stabilise_duration_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_value</code>	(numeric(1)) The lowest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 [lubridate::Period](#) vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-duration>` when `x` cannot be coerced to a duration.
- `<stbl-error-incompatible_values-duration>` when some values cannot be safely converted to a duration.
- `<stbl-error-bad_null>` for `NULL` values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for `NA` values when `allow_na = FALSE`.
- `<stbl-error-outside_range>` when values fall outside `min_value` or `max_value`.
- `<stbl-error-allowed_values>` when the value is not in `allowed_values`.

See Also

Other duration functions: [specify_dur\(\)](#), [stabilize_dur\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_dur_scalar(lubridate::period(day = 1))
stabilize_dur_scalar("P1D")
try(stabilize_dur_scalar(c("P1D", "P2D")))
try(stabilize_dur_scalar(NULL))
stabilize_dur_scalar(NULL, allow_null = TRUE)
```

stabilize_each

Ensure every element of `x` satisfies a single spec

Description

`stabilize_each()` applies `spec` (a `to_*` function, `stabilize_*` function, or `specify_*`() result) to every element of `x` independently, collecting *every* failing location before erroring, rather than stopping at the first failure like [to_each\(\)](#). `stabilise_each()` is a synonym.

Usage

```

stabilize_each(
  x,
  spec,
  ...,
  simplify = TRUE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilise_each(
  x,
  spec,
  ...,
  simplify = TRUE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>spec</code>	(function) A single stabilizer or coercion function, such as a <code>to_*</code> function (to_chr() , etc.), a <code>stabilize_*</code> function (stabilize_chr() , etc.), or a function produced by a <code>specify_*</code> () call (specify_chr() , etc.). Applied independently to each element of <code>x</code> .
<code>...</code>	Arguments passed to methods.
<code>simplify</code>	(logical(1)) Should per-element results be combined into a single atomic vector when possible (every result has size 1 and shares a common type)? If FALSE, a list is always returned.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

`x` with every element coerced by `spec`, simplified to an atomic vector when `simplify = TRUE` and possible, otherwise a list. Errors with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and `<stbl-error-cant_stabilize_each>` when any element fails `spec`; the condition's `locations` element gives the positions that failed.

See Also

Other multiple type functions: [to_each\(\)](#)

Examples

```
stabilize_each(list("1", "2", "3"), stabilize_int)
stabilize_each(list(1L, 2L, 3L), stabilize_chr)

# Elements that can't be simplified into a common type stay a list
stabilize_each(list(1L, "a"), specify_any_of(specify_int(), specify_chr()))

# Reports every failing location, not just the first
try(stabilize_each(list("1", "a", "b"), stabilize_int))
```

stabilize_fct

Coerce to factor with additional checks

Description

Compared to [to_fct\(\)](#), `stabilize_fct()` checks more details, but is slower. `stabilise_fct()`, `stabilize_factor()`, and `stabilise_factor()` are synonyms of `stabilize_fct()`.

Usage

```
stabilize_fct(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  levels = NULL,
  to_na = character(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilize_factor(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  levels = NULL,
  to_na = character(),
```

```

    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
  )

  stabilise_fct(
    x,
    ...,
    allow_null = TRUE,
    allow_na = TRUE,
    min_size = NULL,
    max_size = NULL,
    levels = NULL,
    to_na = character(),
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
  )

  stabilise_factor(
    x,
    ...,
    allow_null = TRUE,
    allow_na = TRUE,
    min_size = NULL,
    max_size = NULL,
    levels = NULL,
    to_na = character(),
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
  )

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using vctrs::vec_size() .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using vctrs::vec_size() .
<code>levels</code>	(character) Expected levels. If NULL (default), the levels will be computed by base::factor() .
<code>to_na</code>	(character) Values to convert to NA.

<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a factor, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-factor>` when `x` cannot be coerced to factor.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-fct_levels>` when values are not present in levels.

See Also

Other factor functions: [are_fct_ish\(\)](#), [specify_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [to\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_fct(letters)
try(stabilize_fct(NULL, allow_null = FALSE))
try(stabilize_fct(c("a", NA), allow_na = FALSE))
try(stabilize_fct(c("a", "b", "c"), min_size = 5))
try(stabilize_fct(c("a", "b", "c"), max_size = 2))
```

`stabilize_fct_scalar` *Coerce to length-1 factor with additional checks*

Description

Checks whether a vector can be coerced to a length-1 factor. `stabilize_fct_scalar()` is optimized to check for length-1 factors (compared to `stabilize_fct()` with `max_size = 1`). `stabilise_fct_scalar`, `stabilize_factor_scalar()`, and `stabilise_factor_scalar` are synonyms of `stabilize_fct_scalar()`.

Usage

```
stabilize_fct_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  levels = NULL,  
  to_na = character(),  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)  
  
stabilize_factor_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  levels = NULL,  
  to_na = character(),  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)  
  
stabilise_fct_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  levels = NULL,  
  to_na = character(),  
  x_arg = caller_arg(x),  
  call = caller_env(),
```

```

    x_class = object_type(x)
  )

  stabilise_factor_scalar(
    x,
    ...,
    allow_null = FALSE,
    allow_zero_length = FALSE,
    allow_na = TRUE,
    levels = NULL,
    to_na = character(),
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
  )

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>levels</code>	(character) Expected levels. If NULL (default), the levels will be computed by <code>base::factor()</code> .
<code>to_na</code>	(character) Values to convert to NA.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 factor, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-factor>` when `x` cannot be coerced to factor.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-fct_levels>` when values are not present in levels.

See Also

Other factor functions: [are_fct-ish\(\)](#), [specify_fct\(\)](#), [stabilize_fct\(\)](#), [to\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_fct_scalar("a")
try(stabilize_fct_scalar(letters))
try(stabilize_fct_scalar("c", levels = c("a", "b")))
```

stabilize_int

Coerce to integer with additional checks

Description

Compared to [to_int\(\)](#), [stabilize_int\(\)](#) checks more details, but is slower. [stabilise_int\(\)](#), [stabilize_integer\(\)](#), and [stabilise_integer\(\)](#) are synonyms of [stabilize_int\(\)](#).

Usage

```
stabilize_int(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL,
  x_arg = caller_arg(x),
```

```
    call = caller_env(),  
    x_class = object_type(x)  
)
```

```
stabilize_integer(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_int(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  unique = FALSE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_integer(  
  x,
```

```

...,
allow_null = TRUE,
allow_na = TRUE,
coerce_character = TRUE,
coerce_factor = TRUE,
min_size = NULL,
max_size = NULL,
unique = FALSE,
min_value = NULL,
max_value = NULL,
exclusive_min_value = NULL,
exclusive_max_value = NULL,
allowed_values = NULL,
multiple_of = NULL,
x_arg = caller_arg(x),
call = caller_env(),
x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>coerce_character</code>	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while as.integer() and as.double() use the integer index of the factor.
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using vctrs::vec_size() .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using vctrs::vec_size() .
<code>unique</code>	(logical(1)) Should all elements in <code>x</code> be distinct?
<code>min_value</code>	(numeric(1)) The lowest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>exclusive_min_value</code>	(numeric(1)) Similar to <code>max_value</code> , but <code>x</code> must be strictly greater than this value (<code>></code> , not <code>>=</code>). NULL (default) values are not checked.
<code>exclusive_max_value</code>	(numeric(1)) Similar to <code>max_value</code> , but <code>x</code> must be strictly less than this value (<code><</code> , not <code><=</code>). NULL (default) values are not checked.

allowed_values	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.
multiple_of	(numeric(1), positive) x must be an integer multiple of this value. NULL (default) skips the check. For doubles, a small relative tolerance is applied to avoid floating-point false negatives.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as an integer vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-integer>` when x cannot be coerced to integer.
- `<stbl-error-incompatible_values-integer>` when some values cannot be safely converted to integer.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-duplicate_elements>` when `unique = TRUE` and duplicates are present.
- `<stbl-error-outside_range>` when values fall outside `min_value`, `max_value`, `exclusive_min_value`, or `exclusive_max_value`.
- `<stbl-error-allowed_values>` when values are not in `allowed_values`.
- `<stbl-error-not_multiple>` when values are not a multiple of `multiple_of`.

See Also

Other integer functions: [are_int-ish\(\)](#), [specify_int\(\)](#), [stabilize_int_scalar\(\)](#), [to\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```

stabilize_int(1:10)
stabilize_int("1")
stabilize_int(1 + 0i)
stabilize_int(NULL)
try(stabilize_int(NULL, allow_null = FALSE))
try(stabilize_int(c(1, NA), allow_na = FALSE))
try(stabilize_int(letters))
try(stabilize_int("1", coerce_character = FALSE))
try(stabilize_int(factor(c("1", "a"))))
try(stabilize_int(factor("1"), coerce_factor = FALSE))
try(stabilize_int(1:10, min_value = 3))
try(stabilize_int(1:10, max_value = 7))
try(stabilize_int(1:10, exclusive_min_value = 3))
try(stabilize_int(1:10, exclusive_max_value = 8))
try(stabilize_int(1:5, allowed_values = c(1L, 2L, 3L)))
try(stabilize_int(1:5, multiple_of = 2))

```

stabilize_int_scalar *Coerce to length-1 integer with additional checks*

Description

Checks whether a vector can be coerced to a length-1 integer vector. `stabilize_int_scalar()` is optimized to check for length-1 integer vectors (compared to `stabilize_int()` with `max_size = 1`). `stabilise_int_scalar`, `stabilize_integer_scalar()`, and `stabilise_integer_scalar` are synonyms of `stabilize_int_scalar()`.

Usage

```

stabilize_int_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  coerce_character = TRUE,
  coerce_factor = TRUE,
  min_value = NULL,
  max_value = NULL,
  exclusive_min_value = NULL,
  exclusive_max_value = NULL,
  allowed_values = NULL,
  multiple_of = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

```
stabilize_integer_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_int_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_value = NULL,  
  max_value = NULL,  
  exclusive_min_value = NULL,  
  exclusive_max_value = NULL,  
  allowed_values = NULL,  
  multiple_of = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_integer_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_value = NULL,
```

```

    max_value = NULL,
    exclusive_min_value = NULL,
    exclusive_max_value = NULL,
    allowed_values = NULL,
    multiple_of = NULL,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>coerce_character</code>	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while as.integer() and as.double() use the integer index of the factor.
<code>min_value</code>	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.
<code>exclusive_min_value</code>	(numeric(1)) Similar to <code>max_value</code> , but x must be strictly greater than this value (>, not >=). NULL (default) values are not checked.
<code>exclusive_max_value</code>	(numeric(1)) Similar to <code>max_value</code> , but x must be strictly less than this value (<, not <=). NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>multiple_of</code>	(numeric(1), positive) x must be an integer multiple of this value. NULL (default) skips the check. For doubles, a small relative tolerance is applied to avoid floating-point false negatives.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.

`x_class` (character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 integer vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-integer>` when `x` cannot be coerced to integer.
- `<stbl-error-incompatible_values-integer>` when some values cannot be safely converted to integer.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-outside_range>` when values fall outside `min_value`, `max_value`, `exclusive_min_value`, or `exclusive_max_value`.
- `<stbl-error-allowed_values>` when the value is not in `allowed_values`.
- `<stbl-error-not_multiple>` when the value is not a multiple of `multiple_of`.

See Also

Other integer functions: [are_int-ish\(\)](#), [specify_int\(\)](#), [stabilize_int\(\)](#), [to\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_int_scalar(1L)
stabilize_int_scalar("1")
try(stabilize_int_scalar(1:10))
try(stabilize_int_scalar(NULL))
stabilize_int_scalar(NULL, allow_null = TRUE)
```

`stabilize_lgl`*Coerce to logical with additional checks*

Description

Compared to `to_lgl()`, `stabilize_lgl()` checks more details, but is slower. `stabilise_lgl()`, `stabilize_logical()`, and `stabilise_logical()` are synonyms of `stabilize_lgl()`.

Usage

```
stabilize_lgl(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  allowed_values = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilize_logical(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  allowed_values = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
stabilise_lgl(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  allowed_values = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)
```

```

)

stabilise_logical(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a logical vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-logical>` when `x` cannot be coerced to logical.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-allowed_values>` when values are not in `allowed_values`.

See Also

Other logical functions: [are_lgl_ish\(\)](#), [specify_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [to\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
stabilize_lgl(c(TRUE, FALSE, TRUE))
stabilize_lgl("true")
stabilize_lgl(NULL)
try(stabilize_lgl(NULL, allow_null = FALSE))
try(stabilize_lgl(c(TRUE, NA), allow_na = FALSE))
try(stabilize_lgl(letters))
try(stabilize_lgl(c(TRUE, FALSE, TRUE), min_size = 5))
try(stabilize_lgl(c(TRUE, FALSE, TRUE), max_size = 2))
try(stabilize_lgl(c(TRUE, FALSE), allowed_values = TRUE))
```

`stabilize_lgl_scalar` *Coerce to length-1 logical with additional checks*

Description

Checks whether a vector can be coerced to a length-1 logical vector. `stabilize_lgl_scalar()` is optimized to check for length-1 logical vectors (compared to [stabilize_lgl\(\)](#) with `max_size = 1`). `stabilise_lgl_scalar`, `stabilize_logical_scalar()`, and `stabilise_logical_scalar` are synonyms of `stabilize_lgl_scalar()`.

Usage

```
stabilize_lgl_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
```

```

)

stabilize_logical_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilise_lgl_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilise_logical_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the

	check. NA values in x are permitted independently of allowed_values, subject to allow_na.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 logical vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-logical>` when x cannot be coerced to logical.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-allowed_values>` when the value is not in `allowed_values`.

See Also

Other logical functions: `are_lgl-ish()`, `specify_lgl()`, `stabilize_lgl()`, `to()`, `to_lgl()`, `to_lgl_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
stabilize_lgl_scalar(TRUE)
stabilize_lgl_scalar("TRUE")
try(stabilize_lgl_scalar(c(TRUE, FALSE, TRUE)))
try(stabilize_lgl_scalar(NULL))
stabilize_lgl_scalar(NULL, allow_null = TRUE)
```

stabilize_lst	<i>Ensure a list meets expectations</i>
---------------	---

Description

`stabilize_lst()` validates the structure and contents of a list. It can check that specific named elements are present and valid, that extra named elements conform to a shared rule, and that unnamed elements conform to a shared rule. `stabilise_lst()`, `stabilize_list()`, and `stabilise_list()` are synonyms of `stabilize_lst()`.

Usage

```
stabilize_lst(  
  .x,  
  ...,  
  .named = NULL,  
  .unnamed = NULL,  
  .allow_duplicate_names = FALSE,  
  .unique = FALSE,  
  .allow_null = TRUE,  
  .allow_zero_length = TRUE,  
  .min_size = NULL,  
  .max_size = NULL,  
  .required = ...names(),  
  .x_arg = caller_arg(.x),  
  .call = caller_env(),  
  .x_class = object_type(.x)  
)
```

```
stabilize_list(  
  .x,  
  ...,  
  .named = NULL,  
  .unnamed = NULL,  
  .allow_duplicate_names = FALSE,  
  .unique = FALSE,  
  .allow_null = TRUE,  
  .allow_zero_length = TRUE,  
  .min_size = NULL,  
  .max_size = NULL,  
  .required = ...names(),  
  .x_arg = caller_arg(.x),  
  .call = caller_env(),  
  .x_class = object_type(.x)  
)
```

```
stabilise_lst(  
  .x,  
  ...,  
  .named = NULL,  
  .unnamed = NULL,  
  .allow_duplicate_names = FALSE,  
  .unique = FALSE,  
  .allow_null = TRUE,  
  .allow_zero_length = TRUE,  
  .min_size = NULL,  
  .max_size = NULL,  
  .required = ...names(),  
  .x_arg = caller_arg(.x),  
  .call = caller_env(),  
  .x_class = object_type(.x)  
)
```

```

.x,
...,
.named = NULL,
.unnamed = NULL,
.allow_duplicate_names = FALSE,
.unique = FALSE,
.allow_null = TRUE,
.allow_zero_length = TRUE,
.min_size = NULL,
.max_size = NULL,
.required = ...names(),
.x_arg = caller_arg(.x),
.call = caller_env(),
.x_class = object_type(.x)
)

stabilise_list(
.x,
...,
.named = NULL,
.unnamed = NULL,
.allow_duplicate_names = FALSE,
.unique = FALSE,
.allow_null = TRUE,
.allow_zero_length = TRUE,
.min_size = NULL,
.max_size = NULL,
.required = ...names(),
.x_arg = caller_arg(.x),
.call = caller_env(),
.x_class = object_type(.x)
)

```

Arguments

<code>.x</code>	The object to stabilize.
<code>...</code>	Named stabilizer functions, such as <code>stabilize_*</code> functions (stabilize_chr() , etc) or functions produced by <code>specify_*</code> () functions (specify_chr() , etc). Each name corresponds to an element in <code>.x</code> , and the function is used to validate that element when present. Whether the element is required (its absence is an error) is controlled by <code>.required</code> .
<code>.named</code>	Controls how named elements of <code>.x</code> that are <i>not</i> explicitly listed in <code>...</code> are handled. One of: • <code>NULL</code> or <code>FALSE</code> (default): any extra named elements cause an error. • <code>TRUE</code>: extra named elements are allowed, unchecked. • A single stabilizer function, such as a <code>stabilize_*</code> function (stabilize_chr(), etc) or a function produced by a <code>specify_*</code>() function (specify_chr(), etc), used to validate every extra named element.

<code>.unnamed</code>	Controls how unnamed elements of <code>.x</code> are handled. One of: • <code>NULL</code> or <code>FALSE</code> (default): any unnamed elements cause an error. • <code>TRUE</code>: unnamed elements are allowed, unchecked. • A single stabilizer function, such as a <code>stabilize_*</code> function (<code>stabilize_chr()</code>, etc) or a function produced by a <code>specify_*</code> function (<code>specify_chr()</code>, etc), used to validate every unnamed element.
<code>.allow_duplicate_names</code>	(logical(1)) Should <code>.x</code> be allowed to have duplicate names? If <code>FALSE</code> (default), an error is thrown when any named element of <code>.x</code> shares a name with another.
<code>.unique</code>	(logical(1)) Should all elements in <code>.x</code> be distinct? If <code>TRUE</code> , duplicated elements are rejected.
<code>.allow_null</code>	(logical(1)) Is <code>NULL</code> an acceptable value?
<code>.allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>.min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>.max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>.required</code>	(character) Names (from <code>...</code>) of elements that must be present in <code>.x</code> . Defaults to all names in <code>...</code> , so every named spec is required unless you opt it out. Named specs <i>not</i> listed here are optional: if absent, no error is raised; if present, they're validated normally. Pass <code>NULL</code> or <code>character()</code> to make every named spec optional. A zero-length <code>.x</code> (such as <code>list()</code>) skips this check when <code>.allow_zero_length = TRUE</code> .
<code>.x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>.call</code>	(environment) The execution environment to mention as the source of error messages.
<code>.x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The validated list, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-bad_null>` for `NULL` values when `.allow_null = FALSE`.
- `<stbl-error-coerce-list>` when `.x` cannot be coerced to a list.
- `<stbl-error-size_too_small>` when the list is shorter than `.min_size`.
- `<stbl-error-size_too_large>` when the list is longer than `.max_size`.
- `<stbl-error-duplicate_elements>` when `.unique = TRUE` and duplicate elements are present.
- `<stbl-error-unnamed_spec>` when any element passed through `...` is unnamed.

- `<stbl-error-missing_element>` when a required named element is absent.
- `<stbl-error-bad_unnamed>` when unnamed elements are present but `.unnamed` is `NULL` or `FALSE`.
- `<stbl-error-bad_named>` when extra named elements are present but `.named` is `NULL` or `FALSE`.
- `<stbl-error-duplicate_names>` when duplicate names are present and `.allow_duplicate_names` = `FALSE`.

See Also

Other list functions: [assert_present\(\)](#), [specify_lst\(\)](#), [to\(\)](#), [to_lst\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
# Basic validation: named required elements
stabilize_lst(
  list(name = "Alice", age = 30L),
  name = specify_chr_scalar(),
  age = specify_int_scalar()
)

# Allow any non-NULL element with assert_present
stabilize_lst(list(data = mtcars), data = assert_present)

# Validate extra named elements via .named
stabilize_lst(
  list(a = 1L, b = 2L, c = 3L),
  .named = specify_int_scalar()
)

# Allow extra named elements unchecked with .named = TRUE
stabilize_lst(list(a = 1L, b = "anything"), .named = TRUE)

# Validate unnamed elements via .unnamed
stabilize_lst(list(1L, 2L, 3L), .unnamed = specify_int_scalar())

# Allow unnamed elements unchecked with .unnamed = TRUE
stabilize_lst(list(1L, "anything"), .unnamed = TRUE)

# NULL is allowed by default
stabilize_lst(NULL)
```

```

try(stabilize_lst(NULL, .allow_null = FALSE))

# Enforce size constraints
try(stabilize_lst(list(a = 1L), .min_size = 2))

# Reject duplicate names by default; opt in to allow them
try(stabilize_lst(list(a = 1L, a = 2L), .named = specify_int_scalar()))
stabilize_lst(
  list(a = 1L, a = 2L),
  .named = specify_int_scalar(),
  .allow_duplicate_names = TRUE
)

# Mark named specs as optional via .required
stabilize_lst(
  list(a = 1L),
  a = specify_int_scalar(),
  b = specify_int_scalar(),
  .required = "a"
)
try(
  stabilize_lst(list(a = 1L), a = specify_int_scalar(), b = specify_int_scalar())
)

```

stabilize_one_of

Try to coerce or validate x as exactly one of several specifications

Description

`stabilize_one_of()` evaluates every function in ... against `x` and requires that exactly one of them succeeds. It returns the result of that single successful function. If zero functions succeed, an informative error that combines the individual failure messages is thrown; if two or more functions succeed, an error naming the specifications that matched is thrown. `stabilise_one_of()` is a synonym.

`to_one_of()` is analogous to `to()`: it tries to coerce `x` to each type given in ... (as a prototype such as `integer()` or `character()`) and requires that exactly one succeeds.

Usage

```

stabilize_one_of(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

stabilise_one_of(

```

```

    x,
    ...,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

to_one_of(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	For <code>stabilize_one_of()</code> : unnamed stabilizer or coercion functions, such as <code>stabilize_*</code> functions (<code>stabilize_chr()</code> , etc.), <code>to_*</code> functions (<code>to_chr()</code> , etc.), functions produced by <code>specify_*</code> () calls (<code>specify_chr()</code> , etc.), or <code>assert_*</code> () functions (such as <code>assert_not()</code>) that return their input unchanged. For <code>to_one_of()</code> : prototype objects (e.g. <code>integer()</code> , <code>character()</code>) that determine the target types to try, passed as the <code>.to</code> argument of <code>to()</code> .
<code>x_arg</code>	(<code>character(1)</code>) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(<code>environment</code>) The execution environment to mention as the source of error messages.
<code>x_class</code>	(<code>character(1)</code>) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

`x` coerced or validated by the single successful function or prototype in `...`, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-empty_specs>` when no functions are supplied in `...`
- `<stbl-error-named_spec>` when any element of `...` is named.
- `<stbl-error-cant_stabilize_one_of>` when zero, or more than one, provided functions succeed.

See Also

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`,

```

stabilize_date_scalar(), stabilize_dbl(), stabilize_dbl_scalar(), stabilize_df(), stabilize_dttm(),
stabilize_dttm_scalar(), stabilize_dur(), stabilize_dur_scalar(), stabilize_fct(),
stabilize_fct_scalar(), stabilize_int(), stabilize_int_scalar(), stabilize_lgl(), stabilize_lgl_scalar(),
stabilize_lst(), stabilize_time(), stabilize_time_scalar(), to_chr(), to_chr_scalar(),
to_date(), to_date_scalar(), to_dbl(), to_dbl_scalar(), to_dttm(), to_dttm_scalar(),
to_dur(), to_dur_scalar(), to_fct(), to_fct_scalar(), to_int(), to_int_scalar(), to_lgl(),
to_lgl_scalar(), to_time(), to_time_scalar()

```

Examples

```

# Returns x unchanged when exactly one function succeeds
stabilize_one_of("a", stabilize_int, stabilize_chr)

# Coerces via the single matching function (1.5 can't become an integer)
stabilize_one_of(1.5, stabilize_int, stabilize_chr)

# Errors as ambiguous: "1" is both int-ish and dbl-ish
try(stabilize_one_of("1", stabilize_int, stabilize_dbl))

# Errors with a combined message when no function succeeds
try(stabilize_one_of(list(1, TRUE, "23", "maybe"), stabilize_lgl, stabilize_int))
# to_one_of() uses prototypes instead of functions
to_one_of("a", integer(), character())

# "FALSE" coerces to logical, but not to integer
to_one_of("FALSE", logical(), integer())

# Errors as ambiguous: "1" coerces to both integer and double
try(to_one_of("1", integer(), double()))

```

stabilize_time	<i>Coerce to time-of-day with additional checks</i>
----------------	---

Description

Compared to [to_time\(\)](#), `stabilize_time()` checks more details, but is slower. `stabilise_time()` is a synonym of `stabilize_time()`.

Usage

```

stabilize_time(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,

```

```

    max_value = NULL,
    allowed_values = NULL,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

stabilise_time(
  x,
  ...,
  allow_null = TRUE,
  allow_na = TRUE,
  min_size = NULL,
  max_size = NULL,
  unique = FALSE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_na</code>	(logical(1)) Are NA values ok?
<code>min_size</code>	(integer(1)) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(integer(1)) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>unique</code>	(logical(1)) Should all elements in <code>x</code> be distinct?
<code>min_value</code>	(numeric(1)) The lowest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>max_value</code>	(numeric(1)) The highest allowed value for <code>x</code> . If NULL (default) values are not checked.
<code>allowed_values</code>	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in <code>x</code> are permitted independently of <code>allowed_values</code> , subject to <code>allow_na</code> .
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.

`x_class` (character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as an `hms::hms()` vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-time>` when `x` cannot be coerced to a time of day.
- `<stbl-error-incompatible_values-time>` when some values cannot be safely converted to a time of day.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-size_too_small>` when the vector is shorter than `min_size`.
- `<stbl-error-size_too_large>` when the vector is longer than `max_size`.
- `<stbl-error-duplicate_elements>` when `unique = TRUE` and duplicates are present.
- `<stbl-error-outside_range>` when values fall outside `min_value` or `max_value`.
- `<stbl-error-allowed_values>` when values are not in `allowed_values`.

See Also

Other time functions: `specify_time()`, `stabilize_time_scalar()`, `to_time()`, `to_time_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
stabilize_time(hms::hms(0, 20, 13))
stabilize_time("13:20:00Z")
stabilize_time(NULL)
try(stabilize_time(NULL, allow_null = FALSE))
try(stabilize_time(c("13:20:00Z", NA), allow_na = FALSE))
try(stabilize_time("13:20:00"))
try(stabilize_time("13:20:00Z", min_value = "18:00:00Z"))
try(stabilize_time("13:20:00Z", max_value = "06:00:00Z"))
```

`stabilize_time_scalar` *Coerce to length-1 time-of-day with additional checks*

Description

Checks whether a vector can be coerced to a length-1 `hms::hms()` vector. `stabilize_time_scalar()` is optimized to check for length-1 time-of-day vectors (compared to `stabilize_time()` with `max_size = 1`). `stabilise_time_scalar` is a synonym of `stabilize_time_scalar()`.

Usage

```
stabilize_time_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

```
stabilise_time_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  allow_na = TRUE,
  min_value = NULL,
  max_value = NULL,
  allowed_values = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?

allow_na	(logical(1)) Are NA values ok?
min_value	(numeric(1)) The lowest allowed value for x. If NULL (default) values are not checked.
max_value	(numeric(1)) The highest allowed value for x. If NULL (default) values are not checked.
allowed_values	A vector of permitted values (coerced to the target type). NULL (default) skips the check. NA values in x are permitted independently of allowed_values, subject to allow_na.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 `hms::hms()` vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-time>` when x cannot be coerced to a time of day.
- `<stbl-error-incompatible_values-time>` when some values cannot be safely converted to a time of day.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.
- `<stbl-error-bad_na>` for NA values when `allow_na = FALSE`.
- `<stbl-error-outside_range>` when values fall outside `min_value` or `max_value`.
- `<stbl-error-allowed_values>` when the value is not in `allowed_values`.

See Also

Other time functions: `specify_time()`, `stabilize_time()`, `to_time()`, `to_time_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
stabilize_time_scalar(hms::hms(0, 20, 13))
stabilize_time_scalar("13:20:00Z")
try(stabilize_time_scalar(c("13:20:00Z", "14:20:00Z")))
try(stabilize_time_scalar(NULL))
stabilize_time_scalar(NULL, allow_null = TRUE)
```

to	<i>Convert a value to a target type</i>
----	---

Description

to() coerces x to the type of . to, dispatching on the class of . to to the appropriate to_*() function.

Usage

```
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'character'
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'double'
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'data.frame'
to(
```

```
x,
.to,
...,
x_arg = caller_arg(x),
call = caller_env(),
x_class = object_type(x)
)

## S3 method for class 'factor'
to(
  x,
  .to,
  ...,
  levels = NULL,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class '`function`'
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'integer'
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'logical'
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

```
## S3 method for class 'list'
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class '`NULL`'
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## Default S3 method:
to(
  x,
  .to,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>.to</code>	A prototype that determines the target type (e.g., <code>integer()</code> , <code>factor(levels = c("a", "b"))</code>).
<code>...</code>	Arguments passed to methods and on to <code>to_*</code> () functions.
<code>x_arg</code>	(<code>character(1)</code>) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(<code>environment</code>) The execution environment to mention as the source of error messages.
<code>x_class</code>	(<code>character(1)</code>) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
<code>levels</code>	(<code>character</code>) The desired factor levels. For factors, a vector's <code>levels</code> play the same role that <code>allowed_values</code> plays for other types: they restrict <code>x</code> to a fixed set of permitted values.

Value

x coerced to the type of .to, or an error condition with classes <stbl-error>, <stbl-condition>, <rlang_error>, <error>, <condition>, and a specific class by failure mode:

- The failure classes documented for the to_*() function corresponding to the class of .to (for example `to_int()` for .to = integer()).
- <stbl-error-coerce-*> when the class of .to has no corresponding to_*() function.

See Also

Other character functions: `are_chr_ish()`, `specify_chr()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `to_chr()`, `to_chr_scalar()`

Other double functions: `are_dbl_ish()`, `specify_dbl()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `to_dbl()`, `to_dbl_scalar()`

Other integer functions: `are_int_ish()`, `specify_int()`, `stabilize_int()`, `stabilize_int_scalar()`, `to_int()`, `to_int_scalar()`

Other logical functions: `are_lgl_ish()`, `specify_lgl()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `to_lgl()`, `to_lgl_scalar()`

Other factor functions: `are_fct_ish()`, `specify_fct()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `to_fct()`, `to_fct_scalar()`

Other function functions: `are_fn_ish()`, `to_fn()`

Other list functions: `assert_present()`, `specify_lst()`, `stabilize_lst()`, `to_lst()`

Other data frame functions: `specify_df()`, `stabilize_df()`, `to_df()`

Examples

```
to(1L, double())
to(1.0, integer())
to(TRUE, character())
to("1", integer())
to(c("a", "b"), factor(levels = c("a", "b", "c")))
to("mean", mean)
```

to_chr

Coerce to character

Description

Checks whether a vector can be coerced to character without losing information, returning it silently if so. Otherwise an informative error message is signaled. `to_character` is a synonym of `to_chr()`.

Usage

```

to_chr(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_character(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Details

This function has two important differences from `base::as.character()`:

- lists and data.frames are *not* coerced to character. In base R, such objects are coerced to character representations of their elements. For example, `as.character(list(1:3))` returns "1:3". In the unlikely event that this is the expected behavior, use `as.character()` instead.
- NULL values can be rejected as part of the call to this function (with `allow_null = FALSE`).

Named functions are converted to their string name. If the function comes from a package namespace, the result is a "pkg::fn" string. For example, `to_chr(mean)` returns "base::mean". Anonymous functions produce an error.

To preserve the original call-site symbol when `to_chr()` is called inside a wrapper function, use the embrace operator `{{ }}`. For example:

```

my_wrapper <- function(fn) {
  to_chr({{ fn }})
}
my_wrapper(mean) # Returns "base::mean"

```

Value

The input as a character vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-character>` when `x` cannot be coerced to character (for example a `data.frame`, or an anonymous function).
- `<stbl-error-incompatible_values-character>` when some elements of a list cannot be converted to character.
- `<stbl-error-bad_null>` for `NULL` values when `allow_null = FALSE`.

See Also

Other character functions: [are_chr_ish\(\)](#), [specify_chr\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [to\(\)](#), [to_chr_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
to_chr("a")
to_chr(letters)
to_chr(1:10)
to_chr(1 + 0i)
to_chr(NULL)
try(to_chr(NULL, allow_null = FALSE))

# Named functions are converted to their string name.
to_chr(mean)
to_chr(base::mean)
try(to_chr(function(x) x))
```

to_chr_scalar	<i>Coerce to length-1 character</i>
---------------	-------------------------------------

Description

Checks whether a vector can be coerced to a length-1 character vector. `to_character_scalar()` is a synonym of `to_chr_scalar()`.

Usage

```

to_chr_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_character_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 character vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-character>` when `x` cannot be coerced to character (for example a `data.frame`, or an anonymous function).
- `<stbl-error-incompatible_values-character>` when some elements of a list cannot be converted to character.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other character functions: `are_chr_ish()`, `specify_chr()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `to()`, `to_chr()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
to_chr_scalar("a")
try(to_chr_scalar(letters))
```

to_date	<i>Coerce to date</i>
---------	-----------------------

Description

Checks whether a vector can be coerced to a `base::Date` without losing information, returning it silently if so. Otherwise an informative error message is signaled.

Usage

```
to_date(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class '`NULL`'
to_date(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())

## S3 method for class 'character'
to_date(
  x,
  ...,
```

```

    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

## S3 method for class 'factor'
to_date(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'POSIXct'
to_date(x, ...)

## S3 method for class 'POSIXlt'
to_date(x, ...)

## S3 method for class 'numeric'
to_date(x, ...)

## S3 method for class 'integer'
to_date(x, ...)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

Details

Character vectors must use the [RFC 3339](#) full-date format ("YYYY-MM-DD"); any other shape (such as "11/13/2018") is rejected (but see [stabilize_date\(\)](#)). `POSIXct` and `POSIXlt` values are truncated to their date component, discarding the time-of-day and timezone offset. Numeric and integer values are treated as the number of days since the Unix epoch ("1970-01-01").

Value

The input as a `base::Date` vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-incompatible_values-date>` when some values cannot be safely converted to date.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other date functions: `locale_datetime_formats()`, `specify_date()`, `stabilize_date()`, `stabilize_date_scalar()`, `to_date_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
to_date(as.Date("2024-01-01"))
to_date("2024-01-01")
to_date(c("2024-01-01", NA))
to_date(0L)
to_date(as.POSIXct("2024-01-01 23:00:00", tz = "UTC"))
to_date(NULL)
try(to_date("11/13/2018"))
try(to_date(c("2024-01-01", "not-a-date")))
```

to_date_scalar	Coerce to length-1 date
----------------	-------------------------

Description

Checks whether a vector can be coerced to a length-1 `base::Date` vector.

Usage

```
to_date_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
```

```

    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
  )

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 `base::Date` vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-incompatible_values-date>` when some values cannot be safely converted to date.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other date functions: `locale_datetime_formats()`, `specify_date()`, `stabilize_date()`, `stabilize_date_scalar()`, `to_date()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
to_date_scalar("2024-01-01")
try(to_date_scalar(c("2024-01-01", "2024-01-02")))
```

to_dbl

*Coerce to double***Description**

Checks whether a vector can be coerced to double without losing information, returning it silently if so. Otherwise an informative error message is signaled. `to_double` is a synonym of `to_dbl()`.

Usage

```
to_dbl(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_double(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class ``NULL``
to_dbl(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())

## S3 method for class 'character'
to_dbl(
  x,
  ...,
  coerce_character = TRUE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'factor'
to_dbl(
  x,
  ...,
```

```

    coerce_factor = TRUE,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
  )

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>coerce_character</code>	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while <code>as.integer()</code> and <code>as.double()</code> use the integer index of the factor.

Value

The input as a double vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-double>` when `x` cannot be coerced to double.
- `<stbl-error-incompatible_values-double>` when some values cannot be safely converted to double.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other double functions: `are_dbl_ish()`, `specify_dbl()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `to()`, `to_dbl_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`,

```
to_dur(), to_dur_scalar(), to_fct(), to_fct_scalar(), to_int(), to_int_scalar(), to_lgl(),
to_lgl_scalar(), to_time(), to_time_scalar()
```

Examples

```
to_dbl(1:10)
to_dbl("1.1")
to_dbl(1 + 0i)
to_dbl(NULL)
try(to_dbl("a"))
try(to_dbl("1.1", coerce_character = FALSE))
```

to_dbl_scalar	<i>Coerce to length-1 double</i>
---------------	----------------------------------

Description

Checks whether a vector can be coerced to a length-1 double vector. `to_double_scalar()` is a synonym of `to_dbl_scalar()`.

Usage

```
to_dbl_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_double_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

allow_zero_length	(logical(1)) Are zero-length vectors acceptable?
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 double vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-double>` when `x` cannot be coerced to double.
- `<stbl-error-incompatible_values-double>` when some values cannot be safely converted to double.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other double functions: [are_dbl-ish\(\)](#), [specify_dbl\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [to\(\)](#), [to_dbl\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
to_dbl_scalar("1.1")
try(to_dbl_scalar(1:10))
```

to_df

*Ensure a data frame meets expectations***Description**

to_df() checks whether an object can be coerced to a data frame, returning it silently if so. Otherwise an informative error message is signaled. to_data_frame() is a synonym of to_df().

Usage

```
to_df(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_data_frame(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class ``NULL``
to_df(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())
```

Arguments

x	The object to stabilize.
...	Arguments passed to <code>base::as.data.frame()</code> or other methods.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
allow_null	(logical(1)) Is NULL an acceptable value?

Value

The object as a data frame, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-data.frame>` when `x` cannot be coerced to a data frame.
- `<stbl-error-bad_named>` when a list has any unnamed elements.
- `<stbl-error-jagged>` when a list has incompatible element lengths.
- `<stbl-error-bad_null>` for `NULL` values when `allow_null = FALSE`.

See Also

Other data frame functions: [specify_df\(\)](#), [stabilize_df\(\)](#), [to\(\)](#)

Examples

```
to_df(mtcars)
to_df(list(name = "Alice", age = 30L))
to_df(NULL)
try(to_df(NULL, allow_null = FALSE))
try(to_df(c("a", "b", "c")))
to_df(letters)
```

to_dttm

Coerce to date-time

Description

Checks whether a vector can be coerced to a [base::POSIXct](#) without losing information, returning it silently if so. Otherwise an informative error message is signaled. `to_datetime()` is a synonym of `to_dttm()`.

Usage

```
to_dttm(
  x,
  ...,
  tz = "UTC",
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_datetime(
  x,
  ...,
  tz = "UTC",
  x_arg = caller_arg(x),
```

```
    call = caller_env(),
    x_class = object_type(x)
)

## S3 method for class 'POSIXct'
to_dttm(x, ..., tz = "UTC", call = caller_env())

## S3 method for class '`NULL`'
to_dttm(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())

## S3 method for class 'character'
to_dttm(
  x,
  ...,
  tz = "UTC",
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'factor'
to_dttm(
  x,
  ...,
  tz = "UTC",
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'POSIXlt'
to_dttm(x, ..., tz = "UTC", call = caller_env())

## S3 method for class 'Date'
to_dttm(x, ..., tz = "UTC", call = caller_env())

## S3 method for class 'numeric'
to_dttm(x, ..., tz = "UTC", call = caller_env())

## S3 method for class 'integer'
to_dttm(x, ..., tz = "UTC", call = caller_env())
```

Arguments

x	The object to stabilize.
...	Arguments passed to methods.
tz	(character(1)) The time zone to normalize x to. Must be "" or a value from OlsonNames() . Defaults to "UTC".

x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
allow_null	(logical(1)) Is NULL an acceptable value?

Details

Character vectors must use the [RFC 3339](#) date-time format ("YYYY-MM-DDTHH:MM:SS", optionally followed by fractional seconds and either "Z" or a numeric offset such as "+05:00"); any other shape is rejected (but see [stabilize_dttm\(\)](#)). A space may be used instead of "T" to separate the date and time. All values are normalized to the time zone named by tz ("UTC" by default); the underlying instant in time is preserved, only its display time zone changes. Numeric and integer values are treated as the number of seconds since the Unix epoch ("1970-01-01 00:00:00 UTC"). [base::Date](#) values are treated as midnight UTC on that date.

Value

The input as a [base::POSIXct](#) vector, or an error condition with classes <stbl-error>, <stbl-condition>, <rlang_error>, <error>, <condition>, and a specific class by failure mode:

- <stbl-error-incompatible_values-datetime> when some values cannot be safely converted to datetime.
- <stbl-error-bad_tz> when tz is not "" or a value from [OlsonNames\(\)](#).
- <stbl-error-bad_null> for NULL values when allow_null = FALSE.

See Also

Other datetime functions: [locale_datetime_formats\(\)](#), [specify_dttm\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [to_dttm_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
to_dttm(as.POSIXct("2024-01-01 12:00:00", tz = "UTC"))
to_dttm("2024-01-01T12:00:00Z")
to_dttm("2024-01-01T12:00:00-05:00")
```

```
to_dttm(c("2024-01-01T12:00:00Z", NA))
to_dttm(0L)
to_dttm(as.Date("2024-01-01"))
to_dttm(NULL)
try(to_dttm("2024-01-01 12:00:00"))
try(to_dttm(c("2024-01-01T12:00:00Z", "not-a-datetime")))
```

to_dttm_scalar	<i>Coerce to length-1 date-time</i>
----------------	-------------------------------------

Description

Checks whether a vector can be coerced to a length-1 [base::POSIXct](#) vector. `to_datetime_scalar()` is a synonym of `to_dttm_scalar()`.

Usage

```
to_dttm_scalar(
  x,
  ...,
  tz = "UTC",
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_datetime_scalar(
  x,
  ...,
  tz = "UTC",
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

x	The object to stabilize.
...	Arguments passed to methods.
tz	(character(1)) The time zone to normalize x to. Must be "" or a value from OlsonNames() . Defaults to "UTC".
allow_null	(logical(1)) Is NULL an acceptable value?

allow_zero_length	(logical(1)) Are zero-length vectors acceptable?
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 [base::POSIXct](#) vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-incompatible_values-datetime>` when some values cannot be safely converted to datetime.
- `<stbl-error-bad_tz>` when `tz` is not `""` or a value from [OlsonNames\(\)](#).
- `<stbl-error-bad_null>` for `NULL` values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other datetime functions: [locale_datetime_formats\(\)](#), [specify_dttm\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [to_dttm\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
to_dttm_scalar("2024-01-01T12:00:00Z")
try(to_dttm_scalar(c(
  "2024-01-01T12:00:00Z",
  "2024-01-02T12:00:00Z"
)))
```

to_dur	<i>Coerce to a duration</i>
--------	-----------------------------

Description

Checks whether a vector can be coerced to a `lubridate::Period` without losing information, returning it silently if so. Otherwise an informative error message is signaled.

Usage

```
to_dur(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_duration(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class '`NULL`'
to_dur(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())

## S3 method for class 'Period'
to_dur(x, ...)

## S3 method for class 'character'
to_dur(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'factor'
to_dur(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
```

```

    x_class = object_type(x)
  )

  ## S3 method for class 'difftime'
  to_dur(x, ..., call = caller_env())

  ## S3 method for class 'numeric'
  to_dur(x, ..., call = caller_env())

  ## S3 method for class 'integer'
  to_dur(x, ..., call = caller_env())

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

Details

Period is chosen as the target class because it preserves the calendar components (years, months, days, hours, minutes, seconds) of an [RFC 3339](#) / [ISO 8601](#) duration string separately, rather than collapsing them into a single number of seconds. That fidelity comes at a cost: years and months have no fixed length, so comparisons (`min_value`, `max_value`, `allowed_values` in [stabilize_dur\(\)](#)) use `lubridate`'s approximate, nominal lengths (a 365.25-day year, a 30.4375-day month) and may not reflect the actual elapsed time implied by a specific calendar date.

Character vectors must use the [RFC 3339](#) duration format: "P" followed by an optional number of years (Y), months (M), and days (D), then an optional "T"-prefixed block of hours (H), minutes (M), and seconds (S) (for example "P3Y6M4DT12H30M5S", "P23DT23H", or "PT1M"); or a stand-alone week form ("P4W"). The date/time form and the week form cannot be mixed, fractional components are not permitted, and "P" or "PT" alone (with no components) are not valid durations. [base::difftime](#) values (including [hms::hms\(\)](#)) are converted directly, preserving their unit. Numeric and integer values are treated as a (fractional) number of seconds and broken into day/hour/minute/second components, with no year or month component, since a number of seconds cannot unambiguously imply a calendar length.

`to_dur()` requires the **lubridate** package.

Value

The input as a [lubridate::Period](#) vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-incompatible_values-duration>` when some values cannot be safely converted to a duration.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other duration functions: [specify_dur\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [to_dur_scalar\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
to_dur(lubridate::period(year = 1, month = 2, day = 3))
to_dur("P3Y6M4DT12H30M5S")
to_dur("PT1M")
to_dur("P4W")
to_dur(c("P1D", NA))
to_dur(3661)
to_dur(as.difftime(90, units = "mins"))
to_dur(NULL)
try(to_dur("P"))
try(to_dur(c("P1D", "not-a-duration")))
```

to_dur_scalar	<i>Coerce to length-1 duration</i>
---------------	------------------------------------

Description

Checks whether a vector can be coerced to a length-1 [lubridate::Period](#) vector.

Usage

```
to_dur_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
```

```

    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
  )

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 [lubridate::Period](#) vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-incompatible_values-duration>` when some values cannot be safely converted to a duration.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other duration functions: [specify_dur\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [to_dur\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_lgl_scalar\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
to_dur_scalar("P1D")
try(to_dur_scalar(c("P1D", "P2D")))
```

to_each	<i>Coerce each element of x with a single spec</i>
---------	--

Description

to_each() applies spec (a to_* function, stabilize_* function, or specify_*() result) to every element of x, and stops at the first element that fails.

Usage

```
to_each(
  x,
  spec,
  ...,
  simplify = TRUE,
  x_arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	The object to stabilize.
spec	(function) A single stabilizer or coercion function, such as a to_* function (to_chr() , etc.), a stabilize_* function (stabilize_chr() , etc.), or a function produced by a specify_*() call (specify_chr() , etc.). Applied independently to each element of x.
...	Arguments passed to methods.
simplify	(logical(1)) Should per-element results be combined into a single atomic vector when possible (every result has size 1 and shares a common type)? If FALSE, a list is always returned.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.

Value

x with every element coerced by spec, simplified to an atomic vector when simplify = TRUE and possible, otherwise a list. Errors with whatever condition spec throws for the first failing element.

See Also

Other multiple type functions: [stabilize_each\(\)](#)

Examples

```
to_each(list("1", "2", "3"), to_int)
to_each(list(1L, 2L, 3L), to_chr)

# Elements that can't be simplified into a common type stay a list
to_each(list(1L, "a"), specify_any_of(specify_int(), specify_chr()))

# Stops at the first element that fails
try(to_each(list("1", "a"), to_int))
```

to_fct

Coerce to factor

Description

Checks whether a vector can be coerced to a factor without losing information, returning it silently if so. Otherwise an informative error message is signaled. `to_factor` is a synonym of `to_fct()`.

Usage

```
to_fct(
  x,
  ...,
  levels = NULL,
  to_na = character(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_factor(
  x,
  ...,
  levels = NULL,
  to_na = character(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class '`NULL`'
to_fct(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>levels</code>	(character) Expected levels. If NULL (default), the levels will be computed by <code>base::factor()</code> .
<code>to_na</code>	(character) Values to convert to NA.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

Details

This function has important differences from `base::as.factor()` and `base::factor()`:

- Values are never silently coerced to NA unless they are explicitly supplied in the `to_na` argument.
- NULL values can be rejected as part of the call to this function (with `allow_null = FALSE`).

Value

The input as a factor, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-factor>` when `x` cannot be coerced to factor.
- `<stbl-error-incompatible_values-factor>` when some elements of a list cannot be converted to factor.
- `<stbl-error-fct_levels>` when values are not present in `levels`.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other factor functions: `are_fct_ish()`, `specify_fct()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `to()`, `to_fct_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
to_fct("a")
to_fct(1:10)
to_fct(NULL)
try(to_fct(letters[1:5], levels = c("a", "c"), to_na = "b"))
```

to_fct_scalar

Coerce to length-1 factor

Description

Checks whether a vector can be coerced to a length-1 factor. `to_fct_scalar()` is optimized to check for length-1 factors (compared to `stabilize_fct()` with `max_size = 1`). `to_factor_scalar()` is a synonym of `to_fct_scalar()`.

Usage

```
to_fct_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  levels = NULL,
  to_na = character(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_factor_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  levels = NULL,
  to_na = character(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

allow_zero_length	(logical(1)) Are zero-length vectors acceptable?
levels	(character) Expected levels. If NULL (default), the levels will be computed by <code>base::factor()</code> .
to_na	(character) Values to convert to NA.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 factor, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-factor>` when `x` cannot be coerced to factor.
- `<stbl-error-incompatible_values-factor>` when some elements of a list cannot be converted to factor.
- `<stbl-error-fct_levels>` when values are not present in levels.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other factor functions: `are_fct-ish()`, `specify_fct()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `to()`, `to_fct()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
to_fct_scalar("a")
try(to_fct_scalar(letters))
```

to_fn	<i>Coerce to a function</i>
-------	-----------------------------

Description

to_fn() coerces x to a function. to_function() is a synonym of to_fn().

Usage

```
to_fn(  
  x,  
  ...,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)  
  
to_function(  
  x,  
  ...,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)  
  
## S3 method for class ``NULL``  
to_fn(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())  
  
## S3 method for class 'character'  
to_fn(  
  x,  
  ...,  
  allow_null = TRUE,  
  definition_env = rlang::global_env(),  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)  
  
## Default S3 method:  
to_fn(  
  x,  
  ...,  
  definition_env = rlang::global_env(),  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)
```

)

Arguments

x	The object to stabilize.
...	Arguments passed to methods.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
allow_null	(logical(1)) Is NULL an acceptable value?
definition_env	(environment) The environment in which to look up function name. Defaults to <code>rlang::global_env()</code> . This argument is ignored when the input is a namespaced string of the form "pkg: :fn", in which case the package namespace is used instead.

Details

Unlike `rlang::as_function()`, `to_fn()` supports namespaced function names such as "pkg: :fn" in the character method. When the input is a length-0 character vector, `to_fn()` returns NULL (subject to `allow_null`). An input of length > 1 is always an error.

Value

A function, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-function>` when x cannot be coerced to a function.
- `<stbl-error-invalid_function_name>` when x is not a syntactically valid function name.
- `<stbl-error-unknown_function>` when x is a syntactically valid name that doesn't resolve to a known function.
- `<stbl-error-non_scalar>` when x has more than one element.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other function functions: `are_fn_ish()`, `to()`

Examples

```
to_fn("mean")
to_fn(~ . + 1)
to_fn(mean)
to_fn("stats::median")
to_fn(NULL)
```

to_int	<i>Coerce to integer</i>
--------	--------------------------

Description

Checks whether a vector can be coerced to integer without losing information, returning it silently if so. Otherwise an informative error message is signaled. `to_integer` is a synonym of `to_int()`.

Usage

```
to_int(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_integer(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class ``NULL``
to_int(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())

## S3 method for class 'character'
to_int(
  x,
  ...,
  coerce_character = TRUE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'factor'
to_int(
  x,
  ...,
  coerce_factor = TRUE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

)

Arguments

x	The object to stabilize.
...	Arguments passed to methods.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
allow_null	(logical(1)) Is NULL an acceptable value?
coerce_character	(logical(1)) Should character vectors such as "1" and "2.0" be considered numeric-ish?
coerce_factor	(logical(1)) Should factors with values such as "1" and "2.0" be considered numeric-ish? Note that this package uses the character value from the factor, while <code>as.integer()</code> and <code>as.double()</code> use the integer index of the factor.

Value

The input as an integer vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-integer>` when x cannot be coerced to integer.
- `<stbl-error-incompatible_values-integer>` when some values cannot be safely converted to integer.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other integer functions: `are_int-ish()`, `specify_int()`, `stabilize_int()`, `stabilize_int_scalar()`, `to()`, `to_int_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```

to_int(1:10)
to_int("1")
to_int(1 + 0i)
to_int(NULL)
try(to_int(c(1, 2, 3.1, 4, 5.2)))
try(to_int("1", coerce_character = FALSE))
try(to_int(c("1", "2", "3.1", "4", "5.2")))

```

to_int_scalar	<i>Coerce to length-1 integer</i>
---------------	-----------------------------------

Description

Checks whether a vector can be coerced to a length-1 integer vector. `to_integer_scalar()` is a synonym of `to_int_scalar()`.

Usage

```

to_int_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_integer_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?

x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
x_class	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 integer vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-integer>` when `x` cannot be coerced to integer.
- `<stbl-error-incompatible_values-integer>` when some values cannot be safely converted to integer.
- `<stbl-error-bad_null>` for `NULL` values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other integer functions: `are_int-ish()`, `specify_int()`, `stabilize_int()`, `stabilize_int_scalar()`, `to()`, `to_int()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
to_int_scalar("1")
try(to_int_scalar(1:10))
```

to_lgl

Coerce to logical

Description

Checks whether a vector can be coerced to logical without losing information, returning it silently if so. Otherwise an informative error message is signaled. `to_logical` is a synonym of `to_lgl()`.

Usage

```

to_lgl(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_logical(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class '`NULL`'
to_lgl(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

Value

The input as a logical vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-logical>` when `x` cannot be coerced to logical.
- `<stbl-error-incompatible_values-logical>` when some values cannot be safely converted to logical.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other logical functions: [are_lgl-ish\(\)](#), [specify_lgl\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [to\(\)](#), [to_lgl_scalar\(\)](#)

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl_scalar()`, `to_time()`, `to_time_scalar()`

Examples

```
to_lgl(TRUE)
to_lgl("TRUE")
to_lgl(1:10)
to_lgl(NULL)
try(to_lgl(NULL, allow_null = FALSE))
try(to_lgl(letters))
try(to_lgl(list(TRUE)))
```

to_lgl_scalar

Coerce to length-1 logical

Description

Checks whether a vector can be coerced to a length-1 logical vector. `to_logical_scalar()` is a synonym of `to_lgl_scalar()`.

Usage

```
to_lgl_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

to_logical_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 logical vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-logical>` when `x` cannot be coerced to logical.
- `<stbl-error-incompatible_values-logical>` when some values cannot be safely converted to logical.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other logical functions: [are_lgl_ish\(\)](#), [specify_lgl\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [to\(\)](#), [to_lgl\(\)](#)

Other stabilization functions: [assert_contains\(\)](#), [assert_not\(\)](#), [assert_present\(\)](#), [stabilize_all_of\(\)](#), [stabilize_any_of\(\)](#), [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_chr_scalar\(\)](#), [stabilize_date\(\)](#), [stabilize_date_scalar\(\)](#), [stabilize_dbl\(\)](#), [stabilize_dbl_scalar\(\)](#), [stabilize_df\(\)](#), [stabilize_dttm\(\)](#), [stabilize_dttm_scalar\(\)](#), [stabilize_dur\(\)](#), [stabilize_dur_scalar\(\)](#), [stabilize_fct\(\)](#), [stabilize_fct_scalar\(\)](#), [stabilize_int\(\)](#), [stabilize_int_scalar\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lgl_scalar\(\)](#), [stabilize_lst\(\)](#), [stabilize_one_of\(\)](#), [stabilize_time\(\)](#), [stabilize_time_scalar\(\)](#), [to_chr\(\)](#), [to_chr_scalar\(\)](#), [to_date\(\)](#), [to_date_scalar\(\)](#), [to_dbl\(\)](#), [to_dbl_scalar\(\)](#), [to_dttm\(\)](#), [to_dttm_scalar\(\)](#), [to_dur\(\)](#), [to_dur_scalar\(\)](#), [to_fct\(\)](#), [to_fct_scalar\(\)](#), [to_int\(\)](#), [to_int_scalar\(\)](#), [to_lgl\(\)](#), [to_time\(\)](#), [to_time_scalar\(\)](#)

Examples

```
to_lgl_scalar("TRUE")
try(to_lgl_scalar(c(TRUE, FALSE)))
```

to_lst	<i>Ensure a list meets expectations</i>
--------	---

Description

to_lst() checks whether an object can be coerced to a list without losing information, returning it silently if so. Otherwise an informative error message is signaled. to_list() is a synonym of to_lst().

Usage

```
to_lst(x, ..., x_arg = caller_arg(x), call = caller_env())

to_list(x, ..., x_arg = caller_arg(x), call = caller_env())

## S3 method for class 'list'
to_lst(x, ..., x_arg = caller_arg(x), call = caller_env())

## Default S3 method:
to_lst(x, ..., x_arg = caller_arg(x), call = caller_env())

## S3 method for class '`NULL`'
to_lst(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())

## S3 method for class '`function`'
to_lst(
  x,
  ...,
  coerce_function = FALSE,
  x_arg = caller_arg(x),
  call = caller_env()
)
```

Arguments

x	The object to stabilize.
...	Arguments passed to <code>base::as.list()</code> or other methods.
x_arg	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
call	(environment) The execution environment to mention as the source of error messages.
allow_null	(logical(1)) Is NULL an acceptable value?
coerce_function	(logical(1)) Should functions be coerced?

Details

This function has important distinctions from `base::as.list()`:

- Functions can be rejected as part of the call to this function (with `coerce_function = FALSE`, the default). If they are allowed, they'll be coerced to a list concatenating their formals and body (as with `base::as.list()`).
- Primitive functions (such as `base::is.na()` or `base::is.list()`) always throw an error, rather than returning `list(NULL)`.
- NULL values can be rejected as part of the call to this function (with `allow_null = FALSE`).

Value

The object as a list, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-coerce-list>` when `x` is a primitive function.
- `<stbl-error-bad_function>` for functions when `coerce_function = FALSE` (the default).
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other list functions: `assert_present()`, `specify_lst()`, `stabilize_lst()`, `to()`

to_time

Coerce to time-of-day

Description

Checks whether a vector can be coerced to an `hms::hms()` time-of-day vector without losing information, returning it silently if so. Otherwise an informative error message is signaled.

Usage

```
to_time(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class '`NULL`'
to_time(x, ..., allow_null = TRUE, x_arg = caller_arg(x), call = caller_env())

## S3 method for class 'character'
to_time(
  x,
```

```
    ...,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

## S3 method for class 'factor'
to_time(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'POSIXct'
to_time(x, ..., call = caller_env())

## S3 method for class 'POSIXlt'
to_time(x, ..., call = caller_env())

## S3 method for class 'numeric'
to_time(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'integer'
to_time(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

## S3 method for class 'difftime'
to_time(
  x,
  ...,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?

Details

Character vectors must use the [RFC 3339](#) full-time format ("HH:MM:SS", optionally followed by fractional seconds and a mandatory offset of either "Z" or a numeric offset such as "+05:00"); any other shape is rejected. The offset is used only to normalize the value to UTC and is not retained: `to_time()` always returns the time-of-day expressed in UTC. `base::POSIXct` and `base::POSIXlt` values are likewise converted to UTC before their time-of-day component is extracted. Numeric and `base::difftime` values are treated as (fractional) seconds since midnight and must resolve to a value in `[0, 86400)`.

`to_time()` requires the **hms** package.

Value

The input as an `hms::hms()` vector, always expressed in UTC, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-incompatible_values-time>` when some values cannot be safely converted to a time of day.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.

See Also

Other time functions: `specify_time()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_time_scalar()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time_scalar()`

Examples

```

to_time(hms::hms(0, 20, 13))
to_time("13:20:00Z")
to_time("13:20:00-05:00")
to_time(c("13:20:00Z", NA))
to_time(3600)
to_time(as.POSIXct("2024-01-01 13:20:00", tz = "UTC"))
to_time(NULL)
try(to_time("13:20:00"))
try(to_time(c("13:20:00Z", "not-a-time")))

```

to_time_scalar	<i>Coerce to length-1 time-of-day</i>
----------------	---------------------------------------

Description

Checks whether a vector can be coerced to a length-1 `hms::hms()` vector.

Usage

```

to_time_scalar(
  x,
  ...,
  allow_null = FALSE,
  allow_zero_length = FALSE,
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)

```

Arguments

<code>x</code>	The object to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(logical(1)) Is NULL an acceptable value?
<code>allow_zero_length</code>	(logical(1)) Are zero-length vectors acceptable?
<code>x_arg</code>	(character(1)) The name of the object being stabilized to use in error messages. The automatic value will work in most cases, or pass it through from higher-level functions to make error messages clearer in unexported functions.
<code>call</code>	(environment) The execution environment to mention as the source of error messages.
<code>x_class</code>	(character(1)) The class name of the object being stabilized to use in error messages. Use this if you remove a special class from the object before checking its coercion, but want the error message to match the original class.

Value

The input as a length-1 `hms::hms()` vector, or an error condition with classes `<stbl-error>`, `<stbl-condition>`, `<rlang_error>`, `<error>`, `<condition>`, and a specific class by failure mode:

- `<stbl-error-incompatible_values-time>` when some values cannot be safely converted to a time of day.
- `<stbl-error-bad_null>` for NULL values when `allow_null = FALSE`.
- `<stbl-error-bad_empty>` for empty vectors when `allow_zero_length = FALSE`.
- `<stbl-error-non_scalar>` for non-scalar vectors.

See Also

Other time functions: `specify_time()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_time()`

Other stabilization functions: `assert_contains()`, `assert_not()`, `assert_present()`, `stabilize_all_of()`, `stabilize_any_of()`, `stabilize_arg()`, `stabilize_chr()`, `stabilize_chr_scalar()`, `stabilize_date()`, `stabilize_date_scalar()`, `stabilize_dbl()`, `stabilize_dbl_scalar()`, `stabilize_df()`, `stabilize_dttm()`, `stabilize_dttm_scalar()`, `stabilize_dur()`, `stabilize_dur_scalar()`, `stabilize_fct()`, `stabilize_fct_scalar()`, `stabilize_int()`, `stabilize_int_scalar()`, `stabilize_lgl()`, `stabilize_lgl_scalar()`, `stabilize_lst()`, `stabilize_one_of()`, `stabilize_time()`, `stabilize_time_scalar()`, `to_chr()`, `to_chr_scalar()`, `to_date()`, `to_date_scalar()`, `to_dbl()`, `to_dbl_scalar()`, `to_dttm()`, `to_dttm_scalar()`, `to_dur()`, `to_dur_scalar()`, `to_fct()`, `to_fct_scalar()`, `to_int()`, `to_int_scalar()`, `to_lgl()`, `to_lgl_scalar()`, `to_time()`

Examples

```
to_time_scalar("13:20:00Z")
try(to_time_scalar(c("13:20:00Z", "14:20:00Z")))
```

Index

- * **character functions**
 - are_chr_ish, 4
 - specify_chr, 31
 - stabilize_chr, 62
 - stabilize_chr_scalar, 65
 - to, 128
 - to_chr, 131
 - to_chr_scalar, 133
- * **check functions**
 - are_chr_ish, 4
 - are_dbl_ish, 5
 - are_fct_ish, 6
 - are_fn_ish, 8
 - are_int_ish, 9
 - are_lgl_ish, 11
- * **data frame functions**
 - specify_df, 38
 - stabilize_df, 81
 - to, 128
 - to_df, 143
- * **date functions**
 - locale_datetime_formats, 22
 - specify_date, 34
 - stabilize_date, 68
 - stabilize_date_scalar, 71
 - to_date, 135
 - to_date_scalar, 137
- * **datetime functions**
 - locale_datetime_formats, 22
 - specify_dttm, 40
 - stabilize_dttm, 85
 - stabilize_dttm_scalar, 88
 - to_dttm, 144
 - to_dttm_scalar, 147
- * **double functions**
 - are_dbl_ish, 5
 - specify_dbl, 35
 - stabilize_dbl, 73
 - stabilize_dbl_scalar, 77
- to, 128
- to_dbl, 139
- to_dbl_scalar, 141
- * **duration functions**
 - specify_dur, 42
 - stabilize_dur, 91
 - stabilize_dur_scalar, 95
 - to_dur, 149
 - to_dur_scalar, 151
- * **factor functions**
 - are_fct_ish, 6
 - specify_fct, 45
 - stabilize_fct, 99
 - stabilize_fct_scalar, 102
 - to, 128
 - to_fct, 154
 - to_fct_scalar, 156
- * **function functions**
 - are_fn_ish, 8
 - to, 128
 - to_fn, 158
- * **integer functions**
 - are_int_ish, 9
 - specify_int, 47
 - stabilize_int, 104
 - stabilize_int_scalar, 108
 - to, 128
 - to_int, 160
 - to_int_scalar, 162
- * **list functions**
 - assert_present, 15
 - specify_lst, 51
 - stabilize_lst, 117
 - to, 128
 - to_lst, 167
- * **logical functions**
 - are_lgl_ish, 11
 - specify_lgl, 50
 - stabilize_lgl, 112

- stabilize_lgl_scalar, 114
 - to, 128
 - to_lgl, 163
 - to_lgl_scalar, 165
- * **multiple type functions**
 - stabilize_each, 97
 - to_each, 153
- * **specification functions**
 - specify_all_of, 30
 - specify_any_of, 31
 - specify_chr, 31
 - specify_date, 34
 - specify_dbl, 35
 - specify_df, 38
 - specify_dttm, 40
 - specify_dur, 42
 - specify_each, 44
 - specify_fct, 45
 - specify_int, 47
 - specify_lgl, 50
 - specify_lst, 51
 - specify_one_of, 54
 - specify_time, 54
- * **stabilization functions**
 - assert_contains, 12
 - assert_not, 14
 - assert_present, 15
 - stabilize_all_of, 56
 - stabilize_any_of, 58
 - stabilize_arg, 60
 - stabilize_chr, 62
 - stabilize_chr_scalar, 65
 - stabilize_date, 68
 - stabilize_date_scalar, 71
 - stabilize_dbl, 73
 - stabilize_dbl_scalar, 77
 - stabilize_df, 81
 - stabilize_dttm, 85
 - stabilize_dttm_scalar, 88
 - stabilize_dur, 91
 - stabilize_dur_scalar, 95
 - stabilize_fct, 99
 - stabilize_fct_scalar, 102
 - stabilize_int, 104
 - stabilize_int_scalar, 108
 - stabilize_lgl, 112
 - stabilize_lgl_scalar, 114
 - stabilize_lst, 117
 - stabilize_one_of, 121
 - stabilize_time, 123
 - stabilize_time_scalar, 126
 - to_chr, 131
 - to_chr_scalar, 133
 - to_date, 135
 - to_date_scalar, 137
 - to_dbl, 139
 - to_dbl_scalar, 141
 - to_dttm, 144
 - to_dttm_scalar, 147
 - to_dur, 149
 - to_dur_scalar, 151
 - to_fct, 154
 - to_fct_scalar, 156
 - to_int, 160
 - to_int_scalar, 162
 - to_lgl, 163
 - to_lgl_scalar, 165
 - to_time, 168
 - to_time_scalar, 171
- * **time functions**
 - specify_time, 54
 - stabilize_time, 123
 - stabilize_time_scalar, 126
 - to_time, 168
 - to_time_scalar, 171
- are_character_ish(are_chr_ish), 4
- are_chr_ish, 4
- are_chr_ish(), 6, 7, 9–11, 33, 64, 67, 131, 133, 135
- are_dbl_ish, 5
- are_dbl_ish(), 4, 7, 9–11, 38, 76, 80, 131, 140, 142
- are_double_ish(are_dbl_ish), 5
- are_factor_ish(are_fct_ish), 6
- are_fct_ish, 6
- are_fct_ish(), 4, 6, 9–11, 47, 101, 104, 131, 155, 157
- are_fn_ish, 8
- are_fn_ish(), 4, 6, 7, 10, 11, 131, 159
- are_function_ish(are_fn_ish), 8
- are_int_ish, 9
- are_int_ish(), 4, 6, 7, 9, 11, 49, 107, 111, 131, 161, 163
- are_integer_ish(are_int_ish), 9
- are_lgl_ish, 11

- `are_lgl_ish()`, [4](#), [6](#), [7](#), [9](#), [10](#), [51](#), [114](#), [116](#), [131](#), [164](#), [166](#)
- `are_logical_ish(are_lgl_ish)`, [11](#)
- `as.double()`, [5](#), [10](#), [37](#), [49](#), [75](#), [79](#), [106](#), [110](#), [140](#), [161](#)
- `as.integer()`, [5](#), [10](#), [37](#), [49](#), [75](#), [79](#), [106](#), [110](#), [140](#), [161](#)
- `assert_contains`, [12](#)
- `assert_contains()`, [15](#), [16](#), [57](#), [59](#), [61](#), [64](#), [68](#), [70](#), [73](#), [76](#), [80](#), [83](#), [88](#), [91](#), [94](#), [97](#), [101](#), [104](#), [107](#), [111](#), [114](#), [116](#), [120](#), [122](#), [125](#), [127](#), [133](#), [135](#), [137](#), [138](#), [140](#), [142](#), [146](#), [148](#), [151](#), [152](#), [155](#), [157](#), [161](#), [163](#), [165](#), [166](#), [170](#), [172](#)
- `assert_not`, [14](#)
- `assert_not()`, [13](#), [16](#), [56–59](#), [61](#), [64](#), [68](#), [70](#), [73](#), [76](#), [80](#), [83](#), [88](#), [91](#), [94](#), [97](#), [101](#), [104](#), [107](#), [111](#), [114](#), [116](#), [120](#), [122](#), [125](#), [127](#), [133](#), [135](#), [137](#), [138](#), [140](#), [142](#), [146](#), [148](#), [151](#), [152](#), [155](#), [157](#), [161](#), [163](#), [165](#), [166](#), [170](#), [172](#)
- `assert_present`, [15](#)
- `assert_present()`, [13](#), [15](#), [53](#), [57](#), [59](#), [61](#), [64](#), [68](#), [70](#), [73](#), [76](#), [80](#), [83](#), [88](#), [91](#), [94](#), [97](#), [101](#), [104](#), [107](#), [111](#), [114](#), [116](#), [120](#), [122](#), [125](#), [127](#), [131](#), [133](#), [135](#), [137](#), [138](#), [140](#), [142](#), [146](#), [148](#), [151](#), [152](#), [155](#), [157](#), [161](#), [163](#), [165](#), [166](#), [168](#), [170](#), [172](#)
- `base::as.character()`, [132](#)
- `base::as.data.frame()`, [143](#)
- `base::as.factor()`, [155](#)
- `base::as.list()`, [167](#), [168](#)
- `base::Date`, [70–72](#), [135](#), [137](#), [138](#), [146](#)
- `base::difftime`, [150](#), [170](#)
- `base::factor()`, [46](#), [100](#), [103](#), [155](#), [157](#)
- `base::is.list()`, [168](#)
- `base::is.na()`, [168](#)
- `base::POSIXct`, [87](#), [88](#), [91](#), [144](#), [146–148](#), [170](#)
- `base::POSIXlt`, [170](#)
- `cli::cli_abort()`, [24](#)
- `cli::cli_bullets()`, [24](#), [25](#), [27](#), [29](#)
- `cli::cli_inform()`, [25](#), [26](#)
- `cli::cli_warn()`, [26](#), [27](#)
- `expect_pkg_error_classes`, [16](#)
- `expect_pkg_error_classes()`, [17](#), [18](#)
- `expect_pkg_error_snapshot`, [17](#)
- `expect_pkg_message_classes`, [18](#)
- `expect_pkg_message_classes()`, [19](#)
- `expect_pkg_message_snapshot`, [19](#)
- `expect_pkg_warning_classes`, [20](#)
- `expect_pkg_warning_classes()`, [21](#)
- `expect_pkg_warning_snapshot`, [21](#)
- `hms::hms()`, [125–127](#), [150](#), [168](#), [170–172](#)
- `ignore_stbl_error`, [22](#)
- `is_character_ish(are_chr_ish)`, [4](#)
- `is_chr_ish(are_chr_ish)`, [4](#)
- `is_dbl_ish(are_dbl_ish)`, [5](#)
- `is_double_ish(are_dbl_ish)`, [5](#)
- `is_factor_ish(are_fct_ish)`, [6](#)
- `is_fct_ish(are_fct_ish)`, [6](#)
- `is_fn_ish(are_fn_ish)`, [8](#)
- `is_function_ish(are_fn_ish)`, [8](#)
- `is_int_ish(are_int_ish)`, [9](#)
- `is_integer_ish(are_int_ish)`, [9](#)
- `is_lgl_ish(are_lgl_ish)`, [11](#)
- `is_logical_ish(are_lgl_ish)`, [11](#)
- `locale_datetime_formats`, [22](#)
- `locale_datetime_formats()`, [35](#), [42](#), [69](#), [70](#), [72](#), [87](#), [88](#), [90](#), [91](#), [137](#), [138](#), [146](#), [148](#)
- `lubridate::Period`, [94](#), [95](#), [97](#), [149](#), [151](#), [152](#)
- `names()`, [28](#)
- `object_type`, [23](#)
- `OlsonNames()`, [41](#), [86](#), [87](#), [90](#), [91](#), [145–148](#)
- `pkg_abort`, [24](#)
- `pkg_abort()`, [16](#)
- `pkg_inform`, [25](#)
- `pkg_inform()`, [18](#)
- `pkg_warn`, [26](#)
- `pkg_warn()`, [20](#)
- `regex_must_match`, [27](#)
- `regex_must_match()`, [33](#), [64](#), [67](#)
- `regex_must_not_match`
(`regex_must_match`), [27](#)
- `replace_stbl_error`, [28](#)
- `rlang::abort()`, [24](#), [26](#), [27](#)
- `rlang::as_function()`, [8](#), [159](#)
- `rlang::global_env()`, [159](#)
- `rlang::inform()`, [26](#)

- `rlang::try_fetch()`, [24, 26, 27](#)
- `rlang::warn()`, [27](#)
- `specify_all_of`, [30](#)
- `specify_all_of()`, [31, 33, 35, 38, 40, 42, 44, 45, 47, 49, 51, 53, 54, 56](#)
- `specify_any_of`, [31](#)
- `specify_any_of()`, [30, 33, 35, 38, 40, 42, 44, 45, 47, 49, 51, 53, 54, 56](#)
- `specify_character(specify_chr)`, [31](#)
- `specify_character_scalar(specify_chr)`, [31](#)
- `specify_chr`, [31](#)
- `specify_chr()`, [4, 12, 30, 31, 35, 38–40, 42, 44, 45, 47, 49, 51–54, 56, 58, 64, 67, 82, 98, 118, 119, 122, 131, 133, 135, 153](#)
- `specify_chr_scalar(specify_chr)`, [31](#)
- `specify_data_frame(specify_df)`, [38](#)
- `specify_date`, [34](#)
- `specify_date()`, [23, 30, 31, 33, 38, 40, 42, 44, 45, 47, 49, 51, 53, 54, 56, 70, 72, 137, 138](#)
- `specify_date_scalar(specify_date)`, [34](#)
- `specify_datetime(specify_dttm)`, [40](#)
- `specify_datetime_scalar(specify_dttm)`, [40](#)
- `specify_dbl`, [35](#)
- `specify_dbl()`, [6, 30, 31, 33, 35, 40, 42, 44, 45, 47, 49, 51, 53, 54, 56, 76, 80, 131, 140, 142](#)
- `specify_dbl_scalar(specify_dbl)`, [35](#)
- `specify_df`, [38](#)
- `specify_df()`, [30, 31, 33, 35, 38, 42, 44, 45, 47, 49, 51, 53, 54, 56, 83, 131, 144](#)
- `specify_double(specify_dbl)`, [35](#)
- `specify_double_scalar(specify_dbl)`, [35](#)
- `specify_dttm`, [40](#)
- `specify_dttm()`, [23, 30, 31, 33, 35, 38, 40, 44, 45, 47, 49, 51, 53, 54, 56, 88, 91, 146, 148](#)
- `specify_dttm_scalar(specify_dttm)`, [40](#)
- `specify_dur`, [42](#)
- `specify_dur()`, [30, 31, 33, 35, 38, 40, 42, 45, 47, 49, 51, 53, 54, 56, 94, 97, 151, 152](#)
- `specify_dur_scalar(specify_dur)`, [42](#)
- `specify_duration(specify_dur)`, [42](#)
- `specify_duration_scalar(specify_dur)`, [42](#)
- `specify_each`, [44](#)
- `specify_each()`, [30, 31, 33, 35, 38, 40, 42, 44, 47, 49, 51, 53, 54, 56](#)
- `specify_factor(specify_fct)`, [45](#)
- `specify_factor_scalar(specify_fct)`, [45](#)
- `specify_fct`, [45](#)
- `specify_fct()`, [7, 30, 31, 33, 35, 38, 40, 42, 44, 45, 49, 51, 53, 54, 56, 101, 104, 131, 155, 157](#)
- `specify_fct_scalar(specify_fct)`, [45](#)
- `specify_int`, [47](#)
- `specify_int()`, [10, 30, 31, 33, 35, 38, 40, 42, 44, 45, 47, 51, 53, 54, 56, 107, 111, 131, 161, 163](#)
- `specify_int_scalar(specify_int)`, [47](#)
- `specify_integer(specify_int)`, [47](#)
- `specify_integer_scalar(specify_int)`, [47](#)
- `specify_lgl`, [50](#)
- `specify_lgl()`, [11, 30, 31, 33, 35, 38, 40, 42, 44, 45, 47, 49, 53, 54, 56, 114, 116, 131, 164, 166](#)
- `specify_lgl_scalar(specify_lgl)`, [50](#)
- `specify_list(specify_lst)`, [51](#)
- `specify_logical(specify_lgl)`, [50](#)
- `specify_logical_scalar(specify_lgl)`, [50](#)
- `specify_lst`, [51](#)
- `specify_lst()`, [16, 30, 31, 33, 35, 38, 40, 42, 44, 45, 47, 49, 51, 54, 56, 120, 131, 168](#)
- `specify_one_of`, [54](#)
- `specify_one_of()`, [30, 31, 33, 35, 38, 40, 42, 44, 45, 47, 49, 51, 53, 56](#)
- `specify_time`, [54](#)
- `specify_time()`, [30, 31, 33, 35, 38, 40, 42, 44, 45, 47, 49, 51, 53, 54, 125, 127, 170, 172](#)
- `specify_time_scalar(specify_time)`, [54](#)
- `stabilise_all_of(stabilize_all_of)`, [56](#)
- `stabilise_any_of(stabilize_any_of)`, [58](#)
- `stabilise_character(stabilize_chr)`, [62](#)
- `stabilise_character_scalar(stabilize_chr_scalar)`, [65](#)
- `stabilise_chr(stabilize_chr)`, [62](#)
- `stabilise_chr_scalar(stabilize_chr_scalar)`, [65](#)
- `stabilise_data_frame(stabilize_df)`, [81](#)

- `stabilise_date (stabilize_date)`, 68
- `stabilise_date_scalar`
 - `(stabilize_date_scalar)`, 71
- `stabilise_datetime (stabilize_dttm)`, 85
- `stabilise_datetime_scalar`
 - `(stabilize_dttm_scalar)`, 88
- `stabilise_dbl (stabilize_dbl)`, 73
- `stabilise_dbl_scalar`
 - `(stabilize_dbl_scalar)`, 77
- `stabilise_df (stabilize_df)`, 81
- `stabilise_double (stabilize_dbl)`, 73
- `stabilise_double_scalar`
 - `(stabilize_dbl_scalar)`, 77
- `stabilise_dttm (stabilize_dttm)`, 85
- `stabilise_dttm_scalar`
 - `(stabilize_dttm_scalar)`, 88
- `stabilise_dur (stabilize_dur)`, 91
- `stabilise_dur_scalar`
 - `(stabilize_dur_scalar)`, 95
- `stabilise_duration (stabilize_dur)`, 91
- `stabilise_duration_scalar`
 - `(stabilize_dur_scalar)`, 95
- `stabilise_each (stabilize_each)`, 97
- `stabilise_factor (stabilize_fct)`, 99
- `stabilise_factor_scalar`
 - `(stabilize_fct_scalar)`, 102
- `stabilise_fct (stabilize_fct)`, 99
- `stabilise_fct_scalar`
 - `(stabilize_fct_scalar)`, 102
- `stabilise_int (stabilize_int)`, 104
- `stabilise_int_scalar`
 - `(stabilize_int_scalar)`, 108
- `stabilise_integer (stabilize_int)`, 104
- `stabilise_integer_scalar`
 - `(stabilize_int_scalar)`, 108
- `stabilise_lgl (stabilize_lgl)`, 112
- `stabilise_lgl_scalar`
 - `(stabilize_lgl_scalar)`, 114
- `stabilise_list (stabilize_lst)`, 117
- `stabilise_logical (stabilize_lgl)`, 112
- `stabilise_logical_scalar`
 - `(stabilize_lgl_scalar)`, 114
- `stabilise_lst (stabilize_lst)`, 117
- `stabilise_one_of (stabilize_one_of)`, 121
- `stabilise_time (stabilize_time)`, 123
- `stabilise_time_scalar`
 - `(stabilize_time_scalar)`, 126
- `stabilize_all_of`, 56
- `stabilize_all_of()`, 13, 15, 16, 30, 59, 61, 64, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 122, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172
- `stabilize_any_of`, 58
- `stabilize_any_of()`, 13, 15, 16, 31, 57, 61, 64, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 122, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172
- `stabilize_arg`, 60
- `stabilize_arg()`, 13, 15, 16, 57, 59, 64, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 122, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172
- `stabilize_arg_scalar (stabilize_arg)`, 60
- `stabilize_arg_scalar()`, 61
- `stabilize_character (stabilize_chr)`, 62
- `stabilize_character_scalar`
 - `(stabilize_chr_scalar)`, 65
- `stabilize_chr`, 62
- `stabilize_chr()`, 4, 12, 13, 15, 16, 22, 27, 29, 31, 33, 39, 45, 52, 56–59, 61, 65, 67, 68, 70, 73, 76, 80, 82, 83, 88, 91, 94, 97, 98, 101, 104, 107, 111, 114, 116, 118–120, 122, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151–153, 155, 157, 161, 163, 165, 166, 170, 172
- `stabilize_chr_scalar`, 65
- `stabilize_chr_scalar()`, 4, 13, 15, 16, 27, 31, 33, 57, 59, 61, 64, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 122, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172
- `stabilize_data_frame (stabilize_df)`, 81
- `stabilize_date`, 68
- `stabilize_date()`, 13, 15, 16, 22, 23, 34, 35, 57, 59, 61, 64, 68, 71–73, 76, 80, 83,

- 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 122, 125, 127, 133, 135–138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_date_scalar`, 71
- `stabilize_date_scalar()`, *13, 15, 16, 23, 34, 35, 57, 59, 61, 64, 68, 70, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_datetime(stabilize_dttm)`, 85
- `stabilize_datetime_scalar(stabilize_dttm_scalar)`, 88
- `stabilize_dbl`, 73
- `stabilize_dbl()`, *6, 13, 15, 16, 35, 38, 57, 59, 61, 64, 68, 70, 73, 77, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_dbl_scalar`, 77
- `stabilize_dbl_scalar()`, *6, 13, 15, 16, 35, 38, 57, 59, 61, 64, 68, 70, 73, 76, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_df`, 81
- `stabilize_df()`, *13, 15, 16, 38, 39, 57, 59, 61, 64, 68, 70, 73, 76, 80, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 144, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_double(stabilize_dbl)`, 73
- `stabilize_double_scalar(stabilize_dbl_scalar)`, 77
- `stabilize_dttm`, 85
- `stabilize_dttm()`, *13, 15, 16, 22, 23, 40, 42, 57, 59, 61, 64, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_dttm_scalar`, 88
- `stabilize_dttm_scalar()`, *13, 15, 16, 23, 40, 42, 57, 59, 61, 64, 68, 70, 73, 76, 80, 83, 88, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_dur`, 91
- `stabilize_dur()`, *13, 15, 16, 42, 44, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 95, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 150–152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_dur_scalar`, 95
- `stabilize_dur_scalar()`, *13, 15, 16, 42, 44, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_duration(stabilize_dur)`, 91
- `stabilize_duration_scalar(stabilize_dur_scalar)`, 95
- `stabilize_each`, 97
- `stabilize_each()`, *44, 45, 154*
- `stabilize_factor(stabilize_fct)`, 99
- `stabilize_factor_scalar(stabilize_fct_scalar)`, 102
- `stabilize_fct`, 99
- `stabilize_fct()`, *7, 13, 15, 16, 45–47, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 102, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155–157, 161, 163, 165, 166, 170, 172*
- `stabilize_fct_scalar`, 102
- `stabilize_fct_scalar()`, *7, 13, 15, 16, 45–47, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 107, 111, 114, 116, 120, 123, 125, 127, 131,*

- 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_int`, 104
- `stabilize_int()`, *10, 13, 15, 16, 47, 49, 57, 59–61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 108, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_int_scalar`, 108
- `stabilize_int_scalar()`, *10, 13, 15, 16, 47, 49, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_integer (stabilize_int)`, 104
- `stabilize_integer_scalar (stabilize_int_scalar)`, 108
- `stabilize_lgl`, 112
- `stabilize_lgl()`, *11, 13, 15, 16, 50, 51, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163–166, 170, 172*
- `stabilize_lgl_scalar`, 114
- `stabilize_lgl_scalar()`, *11, 13, 15, 16, 50, 51, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163–166, 170, 172*
- `stabilize_list (stabilize_lst)`, 117
- `stabilize_logical (stabilize_lgl)`, 112
- `stabilize_logical_scalar (stabilize_lgl_scalar)`, 114
- `stabilize_lst`, 117
- `stabilize_lst()`, *13, 15, 16, 51, 53, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 123, 125, 127, 131, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_one_of`, 121
- `stabilize_one_of()`, *13, 15, 16, 54, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 125, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_time`, 123
- `stabilize_time()`, *13, 15, 16, 54–57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 126, 127, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `stabilize_time_scalar`, 126
- `stabilize_time_scalar()`, *13, 15, 16, 54–57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 133, 135, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `testthat::expect_error()`, 29
- `testthat::expect_snapshot()`, 17–21
- `to`, 128
- `to()`, *4, 6, 7, 9–11, 16, 33, 38, 39, 47, 49, 51, 53, 58, 64, 67, 76, 80, 83, 101, 104, 107, 111, 114, 116, 120–122, 133, 135, 140, 142, 144, 155, 157, 159, 161, 163, 164, 166, 168*
- `to_any_of (stabilize_any_of)`, 58
- `to_character (to_chr)`, 131
- `to_character_scalar (to_chr_scalar)`, 133
- `to_chr`, 131
- `to_chr()`, *4, 12, 13, 15, 16, 33, 45, 56–59, 61, 62, 64, 65, 67, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 98, 101, 104, 107, 111, 114, 116, 120, 122, 123, 125, 127, 131, 135, 137, 138, 140, 142, 146, 148, 151–153, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_chr_scalar`, 133
- `to_chr_scalar()`, *4, 13, 15, 16, 33, 57, 59, 61, 64, 65, 67, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111,*

- 114, 116, 120, 123, 125, 127, 131, 133, 137, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_data_frame(to_df)`, 143
- `to_date`, 135
- `to_date()`, *13, 15, 16, 23, 35, 57, 59, 61, 65, 68, 70, 72, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 138, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_date_scalar`, 137
- `to_date_scalar()`, *13, 15, 16, 23, 35, 57, 59, 61, 65, 68, 70, 72, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 140, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_datetime(to_dttm)`, 144
- `to_datetime_scalar(to_dttm_scalar)`, 147
- `to_dbl`, 139
- `to_dbl()`, *6, 13, 15, 16, 38, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 142, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_dbl_scalar`, 141
- `to_dbl_scalar()`, *6, 13, 15, 16, 38, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 140, 146, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_df`, 143
- `to_df()`, *39, 83, 131*
- `to_double(to_dbl)`, 139
- `to_double_scalar(to_dbl_scalar)`, 141
- `to_dttm`, 144
- `to_dttm()`, *13, 15, 16, 23, 42, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 85, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 140, 142, 148, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_dttm_scalar`, 147
- `to_dttm_scalar()`, *13, 15, 16, 23, 42, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 140, 142, 146, 151, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_dur`, 149
- `to_dur()`, *13, 15, 16, 44, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 141, 142, 146, 148, 152, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_dur_scalar`, 151
- `to_dur_scalar()`, *13, 15, 16, 44, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 133, 135, 137, 138, 141, 142, 146, 148, 151, 155, 157, 161, 163, 165, 166, 170, 172*
- `to_duration(to_dur)`, 149
- `to_each`, 153
- `to_each()`, 97, 99
- `to_factor(to_fct)`, 154
- `to_factor_scalar(to_fct_scalar)`, 156
- `to_fct`, 154
- `to_fct()`, *7, 13, 15, 16, 47, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 99, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 141, 142, 146, 148, 151, 152, 157, 161, 163, 165, 166, 170, 172*
- `to_fct_scalar`, 156
- `to_fct_scalar()`, *7, 13, 15, 16, 47, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120, 123, 125, 127, 131, 133, 135, 137, 138, 141, 142, 146, 148, 151, 152, 155, 161, 163, 165, 166, 170, 172*
- `to_fn`, 158
- `to_fn()`, *8, 9, 131*
- `to_function(to_fn)`, 158
- `to_int`, 160
- `to_int()`, *10, 13, 15, 16, 49, 57, 59, 61, 65, 68, 70, 73, 76, 80, 83, 88, 91, 94, 97, 101, 104, 107, 111, 114, 116, 120,*

123, 125, 127, 131, 133, 135, 137,
138, 141, 142, 146, 148, 151, 152,
155, 157, 163, 165, 166, 170, 172
 to_int_scalar, 162
 to_int_scalar(), *10, 13, 15, 16, 49, 57, 59,*
61, 65, 68, 70, 73, 76, 80, 83, 88, 91,
94, 97, 101, 104, 107, 111, 114, 116,
120, 123, 125, 127, 131, 133, 135,
137, 138, 141, 142, 146, 148, 151,
152, 155, 157, 161, 165, 166, 170,
172
 to_integer(to_int), 160
 to_integer_scalar(to_int_scalar), 162
 to_lgl, 163
 to_lgl(), *11, 13, 15, 16, 51, 57, 59, 61, 65,*
68, 70, 73, 76, 80, 83, 88, 91, 94, 97,
101, 104, 107, 111, 112, 114, 116,
120, 123, 125, 127, 131, 133, 135,
137, 138, 141, 142, 146, 148, 151,
152, 155, 157, 161, 163, 166, 170,
172
 to_lgl_scalar, 165
 to_lgl_scalar(), *11, 13, 15, 16, 51, 57, 59,*
61, 65, 68, 70, 73, 76, 80, 83, 88, 91,
94, 97, 101, 104, 107, 111, 114, 116,
120, 123, 125, 127, 131, 133, 135,
137, 138, 141, 142, 146, 148, 151,
152, 155, 157, 161, 163–165, 170,
172
 to_list(to_lst), 167
 to_logical(to_lgl), 163
 to_logical_scalar(to_lgl_scalar), 165
 to_lst, 167
 to_lst(), *16, 53, 120, 131*
 to_one_of(stabilize_one_of), 121
 to_time, 168
 to_time(), *13, 15, 16, 56, 57, 59, 61, 65, 68,*
70, 73, 76, 80, 83, 88, 91, 94, 97,
101, 104, 107, 111, 114, 116, 120,
123, 125, 127, 133, 135, 137, 138,
141, 142, 146, 148, 151, 152, 155,
157, 161, 163, 165, 166, 172
 to_time_scalar, 171
 to_time_scalar(), *13, 15, 16, 56, 57, 59, 61,*
65, 68, 70, 73, 76, 80, 83, 88, 91, 94,
97, 101, 104, 107, 111, 114, 116,
120, 123, 125, 127, 133, 135, 137,
138, 141, 142, 146, 148, 151, 152,
155, 157, 161, 163, 165, 166, 170
 vctrs::vec_size(), *33, 34, 37, 41, 43, 44,*
46, 49, 51–53, 55, 60, 61, 63, 69, 75,
87, 93, 100, 106, 113, 119, 124