

Package: stbl (via r-universe)

May 30, 2026

Title Stabilize Function Arguments

Version 0.3.0.9002

Description A set of consistent, opinionated functions to quickly check function arguments, 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, rlang (>= 1.0.3), vctrs

Suggests astgrepr, knitr, rmarkdown, stringi, stringr, testthat (>= 3.3.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

NeedsCompilation yes

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Repository <https://wranglezone.r-universe.dev>

Date/Publication 2026-05-30 14:07:14 UTC

RemoteUrl <https://github.com/wranglezone/stbl>

RemoteRef HEAD

RemoteSha dd75c4069e1061d2450483c40b27a808f2037b1e

Contents

are_chr_ish	2
are_dbl_ish	4
are_fct_ish	5
are_int_ish	7
are_lgl_ish	8
expect_pkg_error_classes	9
expect_pkg_error_snapshot	10
expect_pkg_message_classes	11
expect_pkg_message_snapshot	12
expect_pkg_warning_classes	13
expect_pkg_warning_snapshot	14
object_type	15
pkg_abort	15
pkg_inform	16
pkg_warn	18
regex_must_match	19
specify_chr	20
specify_dbl	21
specify_df	23
specify_fct	25
specify_int	27
specify_lgl	29
specify_lst	30
stabilize_arg	32
stabilize_chr	33
stabilize_dbl	38
stabilize_df	44
stabilize_fct	47
stabilize_int	52
stabilize_lgl	57
stabilize_lst	62
stabilize_present	65
to_df	66
to_lst	67

Index 69

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

x	The object to check.
...	Arguments passed to methods.
depth	(length-1 integer) 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\(\)](#)

Other check functions: [are_dbl_ish\(\)](#), [are_fct_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 *Check if an object can be safely coerced to double*

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	(length-1 logical) Should character vectors such as "1" and "2.0" be considered numeric-ish?
coerce_factor	(length-1 logical) 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	(length-1 integer) 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\(\)](#)

Other check functions: [are_chr_ish\(\)](#), [are_fct_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.
to_na	(character) Values to convert to NA.
max_levels	(length-1 numeric) Maximum number of distinct non-NA values allowed across the whole vector after applying to_na.
depth	(length-1 integer) 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\(\)](#)

Other check functions: [are_chr_ish\(\)](#), [are_dbl_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_int_ish	<i>Check if an object can be safely coerced to integer</i>
-------------	--

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

x	The object to check.
...	Arguments passed to methods.
coerce_character	(length-1 logical) Should character vectors such as "1" and "2.0" be considered numeric-ish?
coerce_factor	(length-1 logical) 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.
depth	(length-1 integer) 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\(\)](#)

Other check functions: [are_chr_ish\(\)](#), [are_dbl_ish\(\)](#), [are_fct_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

Check if an object can be safely coerced to logical

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

x	The object to check.
...	Arguments passed to methods.
depth	(length-1 integer) 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\(\)](#)

Other check functions: [are_chr_ish\(\)](#), [are_dbl_ish\(\)](#), [are_fct_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"))
```

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	(length-1 character) 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	(length-1 character) 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. The actual evaluation will occur in a child of this environment, with <code>expect_pkg_error_classes()</code> available even if 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	(length-1 character) 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 message invisibly on success or the message on failure. 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>	(length-1 character) 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. The actual evaluation will occur in a child of this environment, with <code>expect_pkg_message_classes()</code> available even if 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>	(length-1 character) The name of the package to use in classes.
<code>...</code>	(character) Components of the class name, from least-specific to most.

Value

The classes of the warning invisibly on success or the warning on failure. 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>	(length-1 character) 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. The actual evaluation will occur in a child of this environment, with <code>expect_pkg_warning_classes()</code> available even if this package is not attached.

Value

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

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	<i>Signal an error with standards applied</i>
-----------	---

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	(length-1 character) The name of the package to use in classes.
message	(character) The message for the new error. Messages will be formatted with <code>cli::cli_bullets()</code> .
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 <code>rlang::try_fetch()</code> . See <code>rlang::abort()</code> for additional information.
...	Additional parameters passed to <code>cli::cli_abort()</code> and on to <code>rlang::abort()</code> .

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	(length-1 character) 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 <code>rlang::try_fetch()</code> . See <code>rlang::abort()</code> for additional information.
...	Additional parameters passed to <code>cli::cli_inform()</code> and on to <code>rlang::inform()</code> .

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	(length-1 character) 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 argument. 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]")

```

`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,  
  regex = NULL  
)  
  
specify_chr_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  regex = NULL  
)  
  
specify_character(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  regex = NULL  
)  
  
specify_character_scalar(  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  regex = NULL  
)
```

Arguments

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

min_size	(length-1 integer) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
max_size	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
regex	(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.
allow_zero_length	(length-1 logical) 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()`

Other specification functions: `specify_dbl()`, `specify_df()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_lst()`

Examples

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

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,  
  min_value = NULL,  
  max_value = 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  
)  
  
specify_double(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  min_value = NULL,  
  max_value = 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  
)
```

Arguments

allow_null	(length-1 logical) Is NULL an acceptable value?
allow_na	(length-1 logical) Are NA values ok?

coerce_character	(length-1 logical) Should character vectors such as "1" and "2.0" be considered numeric-ish?
coerce_factor	(length-1 logical) 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.
min_size	(length-1 integer) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
max_size	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
min_value	(length-1 numeric) The lowest allowed value for x. If NULL (default) values are not checked.
max_value	(length-1 numeric) The highest allowed value for x. If NULL (default) values are not checked.
allow_zero_length	(length-1 logical) 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()`

Other specification functions: `specify_chr()`, `specify_df()`, `specify_fct()`, `specify_int()`, `specify_lgl()`, `specify_lst()`

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
)

specify_data_frame(
  ...,
  .extra_cols = NULL,
  .col_names = NULL,
  .min_rows = NULL,
  .max_rows = NULL,
  .allow_null = TRUE
)

```

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 required column in <code>.x</code> , and the function is used to validate that column.
<code>.extra_cols</code>	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). This function is used to validate all columns of <code>.x</code> that are <i>not</i> explicitly listed in <code>...</code> . If NULL (default), any extra columns will cause an error.
<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>	(length-1 integer) The minimum number of rows allowed in <code>.x</code> . If NULL (default), the row count is not checked.
<code>.max_rows</code>	(length-1 integer) The maximum number of rows allowed in <code>.x</code> . If NULL (default), the row count is not checked.
<code>.allow_null</code>	(length-1 logical) Is NULL an acceptable value?

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_df()`

Other specification functions: [specify_chr\(\)](#), [specify_dbl\(\)](#), [specify_fct\(\)](#), [specify_int\(\)](#), [specify_lgl\(\)](#), [specify_lst\(\)](#)

Examples

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

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

Other specification functions: `specify_chr()`, `specify_dbl()`, `specify_df()`, `specify_int()`, `specify_lgl()`, `specify_lst()`

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,  
  min_value = NULL,  
  max_value = 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  
)  
  
specify_integer(  
  allow_null = TRUE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  min_value = NULL,  
  max_value = 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
  )

```

Arguments

<code>allow_null</code>	(length-1 logical) Is NULL an acceptable value?
<code>allow_na</code>	(length-1 logical) Are NA values ok?
<code>coerce_character</code>	(length-1 logical) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(length-1 logical) 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>	(length-1 integer) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>min_value</code>	(length-1 numeric) The lowest allowed value for x. If NULL (default) values are not checked.
<code>max_value</code>	(length-1 numeric) The highest allowed value for x. If NULL (default) values are not checked.
<code>allow_zero_length</code>	(length-1 logical) 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()`

Other specification functions: `specify_chr()`, `specify_dbl()`, `specify_df()`, `specify_fct()`, `specify_lgl()`, `specify_lst()`

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

Arguments

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

allow_zero_length

(length-1 logical) 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\(\)](#)

Other specification functions: [specify_chr\(\)](#), [specify_dbl\(\)](#), [specify_df\(\)](#), [specify_fct\(\)](#), [specify_int\(\)](#), [specify_lst\(\)](#)

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,
  .allow_null = TRUE,
  .min_size = NULL,
  .max_size = NULL
)
```

```
specify_list(
  ...,
  .named = NULL,
  .unnamed = NULL,
  .allow_null = TRUE,
  .min_size = NULL,
  .max_size = NULL
)
```

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 required element in <code>.x</code> , and the function is used to validate that element.
<code>.named</code>	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). This function is used to validate all named elements of <code>.x</code> that are <i>not</i> explicitly listed in ... If NULL (default), any extra named elements will cause an error.
<code>.unnamed</code>	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). This function is used to validate all unnamed elements of <code>.x</code> . If NULL (default), any unnamed elements will cause an error.
<code>.allow_null</code>	(length-1 logical) Is NULL an acceptable value?
<code>.min_size</code>	(length-1 integer) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>.max_size</code>	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</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: `stabilize_lst()`, `stabilize_present()`, `to_lst()`

Other specification functions: `specify_chr()`, `specify_dbl()`, `specify_df()`, `specify_fct()`, `specify_int()`, `specify_lgl()`

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"))
  )
)
```

stabilize_arg	<i>Ensure an argument meets expectations</i>
---------------	--

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

x_arg	(length-1 character) The name of the argument 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	(length-1 character) The class name of the argument 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	(length-1 logical) Are zero-length vectors acceptable?

Value

x, unless one of the checks fails.

See Also

Other stabilization functions: [stabilize_chr\(\)](#), [stabilize_dbl\(\)](#), [stabilize_df\(\)](#), [stabilize_fct\(\)](#), [stabilize_int\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lst\(\)](#), [stabilize_present\(\)](#)

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

Ensure a character argument meets expectations

Description

to_chr() checks whether an argument 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().

stabilize_chr() can check more details about the argument, but is slower than to_chr(). stabilise_chr(), stabilize_character(), and stabilise_character() are synonyms of stabilize_chr().

`stabilize_chr_scalar()` and `to_chr_scalar()` are optimized to check for length-1 character vectors. `stabilise_chr_scalar`, `stabilize_character_scalar()`, and `stabilise_character_scalar` are synonyms of `stabilize_chr_scalar()`, and `to_character_scalar()` is a synonym of `to_chr_scalar()`.

Usage

```
stabilize_chr(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  regex = 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,  
  regex = 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,  
  regex = 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,  
    regex = NULL,  
    x_arg = caller_arg(x),  
    call = caller_env(),  
    x_class = object_type(x)  
  )  
  
stabilize_chr_scalar(  
  x,  
  ...,  
  allow_null = FALSE,  
  allow_zero_length = FALSE,  
  allow_na = TRUE,  
  regex = 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,  
  regex = 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,  
  regex = 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,
    regex = NULL,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

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)
)

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

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 argument to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(length-1 logical) Is NULL an acceptable value?
<code>allow_na</code>	(length-1 logical) Are NA values ok?
<code>min_size</code>	(length-1 integer) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<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>x_arg</code>	(length-1 character) The name of the argument 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>	(length-1 character) The class name of the argument 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_zero_length</code>	(length-1 logical) Are zero-length vectors acceptable?

Details

These functions have 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 "1:10". 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`).

Value

The argument as a character vector.

See Also

Other character functions: `are_chr_ish()`, `specify_chr()`

Other stabilization functions: `stabilize_arg()`, `stabilize_dbl()`, `stabilize_df()`, `stabilize_fct()`, `stabilize_int()`, `stabilize_lgl()`, `stabilize_lst()`, `stabilize_present()`

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))

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

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("hide", "find", "find", "hide"), regex = "hide"))

stabilize_chr_scalar(TRUE)
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_dbl

Ensure a double argument meets expectations

Description

`to_dbl()` checks whether an argument 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()`.

`stabilize_dbl()` can check more details about the argument, but is slower than `to_dbl()`. `stabilise_dbl()`, `stabilize_double()`, and `stabilise_double()` are synonyms of `stabilize_dbl()`.

`stabilize_dbl_scalar()` and `to_dbl_scalar()` are optimized to check for length-1 double vectors. `stabilise_dbl_scalar`, `stabilize_double_scalar()`, and `stabilise_double_scalar` are synonyms of `stabilize_dbl_scalar()`, and `to_double_scalar()` is a synonym of `to_dbl_scalar()`.

Usage

```

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

```

```
    coerce_factor = TRUE,  
    min_size = NULL,  
    max_size = NULL,  
    min_value = NULL,  
    max_value = 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,  
  min_value = NULL,  
  max_value = 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,  
  min_value = NULL,  
  max_value = 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,
    min_value = NULL,
    max_value = NULL,
    x_arg = caller_arg(x),
    call = caller_env(),
    x_class = object_type(x)
)

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

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)
)

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 argument to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(length-1 logical) Is NULL an acceptable value?
<code>allow_na</code>	(length-1 logical) Are NA values ok?
<code>coerce_character</code>	(length-1 logical) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(length-1 logical) 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>	(length-1 integer) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .

max_size	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
min_value	(length-1 numeric) The lowest allowed value for x. If NULL (default) values are not checked.
max_value	(length-1 numeric) The highest allowed value for x. If NULL (default) values are not checked.
x_arg	(length-1 character) The name of the argument 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	(length-1 character) The class name of the argument 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	(length-1 logical) Are zero-length vectors acceptable?

Value

The argument as a double vector.

See Also

Other double functions: [are_dbl_ish\(\)](#), [specify_dbl\(\)](#)

Other stabilization functions: [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_df\(\)](#), [stabilize_fct\(\)](#), [stabilize_int\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lst\(\)](#), [stabilize_present\(\)](#)

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

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))

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*Ensure a data frame argument meets expectations*

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

```

Arguments

<code>.x</code>	The argument to stabilize.
<code>...</code>	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 required column in <code>.x</code> , and the function is used to validate that column.
<code>.extra_cols</code>	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). This function is used to validate all columns of <code>.x</code> that are <i>not</i> explicitly listed in <code>...</code> . If <code>NULL</code> (default), any extra columns will cause an error.
<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>	(length-1 integer) 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>	(length-1 integer) 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>	(length-1 logical) Is <code>NULL</code> an acceptable value?
<code>.x_arg</code>	(length-1 character) The name of the argument 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` (length-1 character) The class name of the argument 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.

See Also

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

Other stabilization functions: [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_dbl\(\)](#), [stabilize_fct\(\)](#), [stabilize_int\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lst\(\)](#), [stabilize_present\(\)](#)

Examples

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

# Allow 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 = stabilize_present
)

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

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

```

stabilize_fct

Ensure a factor argument meets expectations

Description

`to_fct()` checks whether an argument 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()`.

`stabilize_fct()` can check more details about the argument, but is slower than `to_fct()`. `stabilise_fct()`, `stabilize_factor()`, and `stabilise_factor()` are synonyms of `stabilize_fct()`.

`stabilize_fct_scalar()` and `to_fct_scalar()` are optimized to check for length-1 factors. `stabilise_fct_scalar`, `stabilize_factor_scalar()`, and `stabilise_factor_scalar` are synonyms of `stabilize_fct_scalar()`, and `to_factor_scalar()` is a synonym of `to_fct_scalar()`.

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)
)

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)
)
```

```
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())

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

to_na	(character) Values to convert to NA.
x_arg	(length-1 character) The name of the argument 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	(length-1 character) The class name of the argument 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	(length-1 logical) Are zero-length vectors acceptable?

Details

These functions have 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 argument as a factor.

See Also

Other factor functions: `are_fct_ish()`, `specify_fct()`

Other stabilization functions: `stabilize_arg()`, `stabilize_chr()`, `stabilize_dbl()`, `stabilize_df()`, `stabilize_int()`, `stabilize_lgl()`, `stabilize_lst()`, `stabilize_present()`

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("a")
try(to_fct_scalar(letters))
```

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

`stabilize_int`*Ensure an integer argument meets expectations*

Description

`to_int()` checks whether an argument 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()`.

`stabilize_int()` can check more details about the argument, but is slower than `to_int()`. `stabilise_int()`, `stabilize_integer()`, and `stabilise_integer()` are synonyms of `stabilize_int()`.

`stabilize_int_scalar()` and `to_int_scalar()` are optimized to check for length-1 integer vectors. `stabilise_int_scalar`, `stabilize_integer_scalar()`, and `stabilise_integer_scalar` are synonyms of `stabilize_int_scalar()`, and `to_integer_scalar()` is a synonym of `to_int_scalar()`.

Usage

```
stabilize_int(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  coerce_character = TRUE,  
  coerce_factor = TRUE,  
  min_size = NULL,  
  max_size = NULL,  
  min_value = NULL,  
  max_value = 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,  
  min_value = NULL,  
  max_value = 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,  
  min_value = NULL,  
  max_value = 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,  
  min_value = NULL,  
  max_value = NULL,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)  
  
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,  
  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,  
  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,  
  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,  
  x_arg = caller_arg(x),  
  call = caller_env(),  
  x_class = object_type(x)  
)
```

```
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)
)

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 argument to stabilize.
<code>...</code>	Arguments passed to methods.
<code>allow_null</code>	(length-1 logical) Is NULL an acceptable value?
<code>allow_na</code>	(length-1 logical) Are NA values ok?
<code>coerce_character</code>	(length-1 logical) Should character vectors such as "1" and "2.0" be considered numeric-ish?
<code>coerce_factor</code>	(length-1 logical) 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>	(length-1 integer) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>max_size</code>	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>min_value</code>	(length-1 numeric) The lowest allowed value for x. If NULL (default) values are not checked.
<code>max_value</code>	(length-1 numeric) The highest allowed value for x. If NULL (default) values are not checked.
<code>x_arg</code>	(length-1 character) The name of the argument 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>	(length-1 character) The class name of the argument 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_zero_length</code>	(length-1 logical) Are zero-length vectors acceptable?

Value

The argument as an integer vector.

See Also

Other integer functions: [are_int-ish\(\)](#), [specify_int\(\)](#)

Other stabilization functions: [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_dbl\(\)](#), [stabilize_df\(\)](#), [stabilize_fct\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lst\(\)](#), [stabilize_present\(\)](#)

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("1")
try(to_int_scalar(1:10))

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))

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

Ensure a logical argument meets expectations

Description

`to_lgl()` checks whether an argument 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()`.

`stabilize_lgl()` can check more details about the argument, but is slower than `to_lgl()`. `stabilise_lgl()`, `stabilize_logical()`, and `stabilise_logical()` are synonyms of `stabilize_lgl()`.

`stabilize_lgl_scalar()` and `to_lgl_scalar()` are optimized to check for length-1 logical vectors. `stabilise_lgl_scalar()`, `stabilize_logical_scalar()`, and `stabilise_logical_scalar()` are synonyms of `stabilize_lgl_scalar()`, and `to_logical_scalar()` is a synonym of `to_lgl_scalar()`.

Usage

```
stabilize_lgl(  
  x,  
  ...,  
  allow_null = TRUE,  
  allow_na = TRUE,  
  min_size = NULL,  
  max_size = 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,  
  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,  
  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,
```

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

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

```

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())

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

max_size	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
x_arg	(length-1 character) The name of the argument 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	(length-1 character) The class name of the argument 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	(length-1 logical) Are zero-length vectors acceptable?

Value

The argument as a logical vector.

See Also

Other logical functions: [are_lgl-ish\(\)](#), [specify_lgl\(\)](#)

Other stabilization functions: [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_dbl\(\)](#), [stabilize_df\(\)](#), [stabilize_fct\(\)](#), [stabilize_int\(\)](#), [stabilize_lst\(\)](#), [stabilize_present\(\)](#)

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("TRUE")
try(to_lgl_scalar(c(TRUE, FALSE)))

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))

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`*Ensure a list argument meets expectations*

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

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

```
stabilise_lst(  
  .x,  
  ...,  
  .named = NULL,  
  .unnamed = NULL,  
  .allow_duplicate_names = FALSE,  
  .allow_null = TRUE,
```

```

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

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

```

Arguments

<code>.x</code>	The argument to stabilize.
<code>...</code>	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 required element in <code>.x</code> , and the function is used to validate that element.
<code>.named</code>	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). This function is used to validate all named elements of <code>.x</code> that are <i>not</i> explicitly listed in <code>...</code> . If NULL (default), any extra named elements will cause an error.
<code>.unnamed</code>	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). This function is used to validate all unnamed elements of <code>.x</code> . If NULL (default), any unnamed elements will cause an error.
<code>.allow_duplicate_names</code>	(length-1 logical) Should <code>.x</code> be allowed to have duplicate names? If FALSE (default), an error is thrown when any named element of <code>.x</code> shares a name with another.
<code>.allow_null</code>	(length-1 logical) Is NULL an acceptable value?
<code>.min_size</code>	(length-1 integer) The minimum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>.max_size</code>	(length-1 integer) The maximum size of the object. Object size will be tested using <code>vctrs::vec_size()</code> .
<code>.x_arg</code>	(length-1 character) The name of the argument 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>	(length-1 character) The class name of the argument 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.

See Also

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

Other stabilization functions: [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_dbl\(\)](#), [stabilize_df\(\)](#), [stabilize_fct\(\)](#), [stabilize_int\(\)](#), [stabilize_lgl\(\)](#), [stabilize_present\(\)](#)

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 stabilize_present
stabilize_lst(list(data = mtcars), data = stabilize_present)

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

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

# 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
  )

```

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

Description

`stabilize_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

```

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

```

Arguments

<code>x</code>	The argument to stabilize.
<code>x_arg</code>	(length-1 character) The name of the argument 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>	(length-1 character) The class name of the argument 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_lst\(\)](#)

Other stabilization functions: [stabilize_arg\(\)](#), [stabilize_chr\(\)](#), [stabilize_dbl\(\)](#), [stabilize_df\(\)](#), [stabilize_fct\(\)](#), [stabilize_int\(\)](#), [stabilize_lgl\(\)](#), [stabilize_lst\(\)](#)

Examples

```

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

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

```

to_df

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

to_df() checks whether an argument 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 argument to stabilize.
...	Arguments passed to <code>base::as.data.frame()</code> or other methods.
x_arg	(length-1 character) The name of the argument 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` (length-1 character) The class name of the argument 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` (length-1 logical) Is NULL an acceptable value?

Value

The argument as a data frame.

See Also

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

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_lst

Ensure a list argument meets expectations

Description

`to_lst()` checks whether an argument can be coerced to a list without losing information, returning it silently if so. Otherwise an informative error message is signaled. `to_lst()` is a synonym of `to_lst()`.

Usage

```
to_lst(x, ..., x_arg = caller_arg(x), call = caller_env())

to_lst(x, ..., x_arg = caller_arg(x), call = caller_env())

## S3 method for class 'list'
to_lst(x, ...)

## 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 argument to stabilize.
...	Arguments passed to <code>base::as.list()</code> or other methods.
x_arg	(length-1 character) The name of the argument 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	(length-1 logical) Is NULL an acceptable value?
coerce_function	(length-1 logical) 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 argument as a list.

See Also

Other list functions: `specify_lst()`, `stabilize_lst()`, `stabilize_present()`

Index

- * **character functions**
 - are_chr_ish, 2
 - specify_chr, 20
 - stabilize_chr, 33
- * **check functions**
 - are_chr_ish, 2
 - are_dbl_ish, 4
 - are_fct_ish, 5
 - are_int_ish, 7
 - are_lgl_ish, 8
- * **data frame functions**
 - specify_df, 23
 - stabilize_df, 44
 - to_df, 66
- * **double functions**
 - are_dbl_ish, 4
 - specify_dbl, 21
 - stabilize_dbl, 38
- * **factor functions**
 - are_fct_ish, 5
 - specify_fct, 25
 - stabilize_fct, 47
- * **integer functions**
 - are_int_ish, 7
 - specify_int, 27
 - stabilize_int, 52
- * **list functions**
 - specify_lst, 30
 - stabilize_lst, 62
 - stabilize_present, 65
 - to_lst, 67
- * **logical functions**
 - are_lgl_ish, 8
 - specify_lgl, 29
 - stabilize_lgl, 57
- * **specification functions**
 - specify_chr, 20
 - specify_dbl, 21
 - specify_df, 23
 - specify_fct, 25
 - specify_int, 27
 - specify_lgl, 29
 - specify_lst, 30
- * **stabilization functions**
 - stabilize_arg, 32
 - stabilize_chr, 33
 - stabilize_dbl, 38
 - stabilize_df, 44
 - stabilize_fct, 47
 - stabilize_int, 52
 - stabilize_lgl, 57
 - stabilize_lst, 62
 - stabilize_present, 65
- are_character_ish (are_chr_ish), 2
- are_chr_ish, 2
- are_chr_ish(), 5, 6, 8, 9, 21, 37
- are_dbl_ish, 4
- are_dbl_ish(), 3, 6, 8, 9, 23, 43
- are_double_ish (are_dbl_ish), 4
- are_factor_ish (are_fct_ish), 5
- are_fct_ish, 5
- are_fct_ish(), 3, 5, 8, 9, 26, 51
- are_int_ish, 7
- are_int_ish(), 3, 5, 6, 9, 28, 57
- are_integer_ish (are_int_ish), 7
- are_lgl_ish, 8
- are_lgl_ish(), 3, 5, 6, 8, 30, 61
- are_logical_ish (are_lgl_ish), 8
- as.double(), 4, 7, 23, 28, 42, 56
- as.integer(), 4, 7, 23, 28, 42, 56
- base::as.character(), 37
- base::as.data.frame(), 66
- base::as.factor(), 51
- base::as.list(), 68
- base::factor(), 26, 50, 51
- base::is.list(), 68
- base::is.na(), 68

- cli::cli_abort(), [15](#), [16](#)
- cli::cli_bullets(), [16–18](#)
- cli::cli_inform(), [16](#), [17](#)
- cli::cli_warn(), [18](#)

- expect_pkg_error_classes, [9](#)
- expect_pkg_error_classes(), [10](#), [11](#)
- expect_pkg_error_snapshot, [10](#)
- expect_pkg_message_classes, [11](#)
- expect_pkg_message_classes(), [12](#)
- expect_pkg_message_snapshot, [12](#)
- expect_pkg_warning_classes, [13](#)
- expect_pkg_warning_classes(), [14](#)
- expect_pkg_warning_snapshot, [14](#)

- is_character_ish(are_chr_ish), [2](#)
- is_chr_ish(are_chr_ish), [2](#)
- is_dbl_ish(are_dbl_ish), [4](#)
- is_double_ish(are_dbl_ish), [4](#)
- is_factor_ish(are_fct_ish), [5](#)
- is_fct_ish(are_fct_ish), [5](#)
- is_int_ish(are_int_ish), [7](#)
- is_integer_ish(are_int_ish), [7](#)
- is_lgl_ish(are_lgl_ish), [8](#)
- is_logical_ish(are_lgl_ish), [8](#)

- names(), [19](#)

- object_type, [15](#)

- pkg_abort, [15](#)
- pkg_abort(), [9](#)
- pkg_inform, [16](#)
- pkg_inform(), [11](#)
- pkg_warn, [18](#)
- pkg_warn(), [13](#)

- regex_must_match, [19](#)
- regex_must_match(), [21](#), [37](#)
- regex_must_not_match
(regex_must_match), [19](#)
- rlang::abort(), [16–18](#)
- rlang::inform(), [17](#)
- rlang::try_fetch(), [16–18](#)
- rlang::warn(), [18](#)

- specify_character(specify_chr), [20](#)
- specify_character_scalar(specify_chr),
[20](#)
- specify_chr, [20](#)
- specify_chr(), [3](#), [23–26](#), [28](#), [30](#), [31](#), [37](#), [45](#),
[63](#)
- specify_chr_scalar(specify_chr), [20](#)
- specify_data_frame(specify_df), [23](#)
- specify_dbl, [21](#)
- specify_dbl(), [5](#), [21](#), [25](#), [26](#), [28](#), [30](#), [31](#), [43](#)
- specify_dbl_scalar(specify_dbl), [21](#)
- specify_df, [23](#)
- specify_df(), [21](#), [23](#), [26](#), [28](#), [30](#), [31](#), [46](#), [67](#)
- specify_double(specify_dbl), [21](#)
- specify_double_scalar(specify_dbl), [21](#)
- specify_factor(specify_fct), [25](#)
- specify_factor_scalar(specify_fct), [25](#)
- specify_fct, [25](#)
- specify_fct(), [6](#), [21](#), [23](#), [25](#), [28](#), [30](#), [31](#), [51](#)
- specify_fct_scalar(specify_fct), [25](#)
- specify_int, [27](#)
- specify_int(), [8](#), [21](#), [23](#), [25](#), [26](#), [30](#), [31](#), [57](#)
- specify_int_scalar(specify_int), [27](#)
- specify_integer(specify_int), [27](#)
- specify_integer_scalar(specify_int), [27](#)
- specify_lgl, [29](#)
- specify_lgl(), [9](#), [21](#), [23](#), [25](#), [26](#), [28](#), [31](#), [61](#)
- specify_lgl_scalar(specify_lgl), [29](#)
- specify_list(specify_lst), [30](#)
- specify_logical(specify_lgl), [29](#)
- specify_logical_scalar(specify_lgl), [29](#)
- specify_lst, [30](#)
- specify_lst(), [21](#), [23](#), [25](#), [26](#), [28](#), [30](#), [64](#), [65](#),
[68](#)
- stabilise_character(stabilize_chr), [33](#)
- stabilise_character_scalar
(stabilize_chr), [33](#)
- stabilise_chr(stabilize_chr), [33](#)
- stabilise_chr_scalar(stabilize_chr), [33](#)
- stabilise_data_frame(stabilize_df), [44](#)
- stabilise_dbl(stabilize_dbl), [38](#)
- stabilise_dbl_scalar(stabilize_dbl), [38](#)
- stabilise_df(stabilize_df), [44](#)
- stabilise_double(stabilize_dbl), [38](#)
- stabilise_double_scalar
(stabilize_dbl), [38](#)
- stabilise_factor(stabilize_fct), [47](#)
- stabilise_factor_scalar
(stabilize_fct), [47](#)
- stabilise_fct(stabilize_fct), [47](#)
- stabilise_fct_scalar(stabilize_fct), [47](#)
- stabilise_int(stabilize_int), [52](#)

- stabilise_int_scalar (stabilize_int), 52
- stabilise_integer (stabilize_int), 52
- stabilise_integer_scalar
(stabilize_int), 52
- stabilise_lgl (stabilize_lgl), 57
- stabilise_lgl_scalar (stabilize_lgl), 57
- stabilise_list (stabilize_lst), 62
- stabilise_logical (stabilize_lgl), 57
- stabilise_logical_scalar
(stabilize_lgl), 57
- stabilise_lst (stabilize_lst), 62
- stabilize_arg, 32
- stabilize_arg(), 37, 43, 46, 51, 57, 61, 64,
65
- stabilize_arg_scalar (stabilize_arg), 32
- stabilize_character (stabilize_chr), 33
- stabilize_character_scalar
(stabilize_chr), 33
- stabilize_chr, 33
- stabilize_chr(), 3, 19–21, 24, 31, 33, 43,
45, 46, 51, 57, 61, 63–65
- stabilize_chr_scalar (stabilize_chr), 33
- stabilize_chr_scalar(), 19–21
- stabilize_data_frame (stabilize_df), 44
- stabilize_dbl, 38
- stabilize_dbl(), 5, 21, 23, 33, 37, 46, 51,
57, 61, 64, 65
- stabilize_dbl_scalar (stabilize_dbl), 38
- stabilize_dbl_scalar(), 21, 23
- stabilize_df, 44
- stabilize_df(), 23, 24, 33, 37, 43, 51, 57,
61, 64, 65, 67
- stabilize_double (stabilize_dbl), 38
- stabilize_double_scalar
(stabilize_dbl), 38
- stabilize_factor (stabilize_fct), 47
- stabilize_factor_scalar
(stabilize_fct), 47
- stabilize_fct, 47
- stabilize_fct(), 6, 25, 26, 33, 37, 43, 46,
57, 61, 64, 65
- stabilize_fct_scalar (stabilize_fct), 47
- stabilize_fct_scalar(), 25, 26
- stabilize_int, 52
- stabilize_int(), 8, 27, 28, 32, 33, 37, 43,
46, 51, 61, 64, 65
- stabilize_int_scalar (stabilize_int), 52
- stabilize_int_scalar(), 27, 28
- stabilize_integer (stabilize_int), 52
- stabilize_integer_scalar
(stabilize_int), 52
- stabilize_lgl, 57
- stabilize_lgl(), 9, 29, 30, 33, 37, 43, 46,
51, 57, 64, 65
- stabilize_lgl_scalar (stabilize_lgl), 57
- stabilize_lgl_scalar(), 29, 30
- stabilize_list (stabilize_lst), 62
- stabilize_logical (stabilize_lgl), 57
- stabilize_logical_scalar
(stabilize_lgl), 57
- stabilize_lst, 62
- stabilize_lst(), 30, 31, 33, 37, 43, 46, 51,
57, 61, 65, 68
- stabilize_present, 65
- stabilize_present(), 31, 33, 37, 43, 46, 51,
57, 61, 64, 68
- testthat::expect_snapshot(), 10–14
- to_character (stabilize_chr), 33
- to_character_scalar (stabilize_chr), 33
- to_chr (stabilize_chr), 33
- to_chr.NULL (stabilize_chr), 33
- to_chr_scalar (stabilize_chr), 33
- to_data_frame (to_df), 66
- to_dbl (stabilize_dbl), 38
- to_dbl.character (stabilize_dbl), 38
- to_dbl.factor (stabilize_dbl), 38
- to_dbl.NULL (stabilize_dbl), 38
- to_dbl_scalar (stabilize_dbl), 38
- to_df, 66
- to_df(), 24, 46
- to_double (stabilize_dbl), 38
- to_double_scalar (stabilize_dbl), 38
- to_factor (stabilize_fct), 47
- to_factor_scalar (stabilize_fct), 47
- to_fct (stabilize_fct), 47
- to_fct.NULL (stabilize_fct), 47
- to_fct_scalar (stabilize_fct), 47
- to_int (stabilize_int), 52
- to_int.character (stabilize_int), 52
- to_int.factor (stabilize_int), 52
- to_int.NULL (stabilize_int), 52
- to_int_scalar (stabilize_int), 52
- to_integer (stabilize_int), 52
- to_integer_scalar (stabilize_int), 52
- to_lgl (stabilize_lgl), 57
- to_lgl.NULL (stabilize_lgl), 57

`to_lgl_scalar` (`stabilize_lgl`), [57](#)
`to_list` (`to_lst`), [67](#)
`to_logical` (`stabilize_lgl`), [57](#)
`to_logical_scalar` (`stabilize_lgl`), [57](#)
`to_lst`, [67](#)
`to_lst()`, [31](#), [64](#), [65](#)

`vctrs::vec_size()`, [21](#), [23](#), [26](#), [28](#), [29](#), [31](#),
[32](#), [37](#), [42](#), [43](#), [50](#), [56](#), [60](#), [61](#), [63](#)