Package 'stbl'

September 16, 2025

Title Stabilize Function Arguments

Version 0.2.0

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.api2r.org/, https://github.com/api2r/stbl

BugReports https://github.com/api2r/stbl/issues

Depends R (>= 4.1)

Imports cli, glue, rlang (>= 1.1.0), vctrs

Suggests knitr, rmarkdown, stringi, stringr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

NeedsCompilation no

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

Maintainer Jon Harmon <jonthegeek@gmail.com>

Repository CRAN

Date/Publication 2025-09-16 21:00:02 UTC

Contents

`are_chr_ish` *Check if an object can be safely coerced to character*

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.

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

Usage

```
are_chr_ish(x, ...)

is_chr_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.

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.

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

Usage

```
are_dbl_ish(x, ...)

is_dbl_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.

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.

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

Usage

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

is_fct_ish(x, ...)

## 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.
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.

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")

# 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 *Check if an object can be safely coerced to integer*

Description

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

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

Usage

```
are_int_ish(x, ...)

is_int_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 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_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.

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.

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

Usage

```
are_lgl_ish(x, ...)

is_lgl_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.

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"))
```

`object_type` *Identify the class, type, etc of an object*

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))
```

`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`, 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)
```

Arguments

regex	(character) The regular expression pattern.

Value

The regex value with names() equal to the generated error message.

Examples

```
regex_must_match("[aeiou]")

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

---

regex_must_not_match *Create a 'must not match' regex rule*

Description

Attach a standardized error message to a regex argument that specifies that the pattern must *not* be matched. This is a wrapper around regex_must_match() that sets the negate attribute to TRUE.

Usage

```
regex_must_not_match(regex)
```

Arguments

regex	(character) The regular expression pattern.

Value

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

Examples

```
regex_must_not_match("[aeiou]")
```

`stabilize_arg` *Ensure an argument meets expectations*

Description

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

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

x_arg (length-1 character) An argument name for x. 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 x to use in error messages. Use this if you remove a special class from x 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.

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.

stabilize_chr() can check more details about the argument, but is slower than to_chr().

stabilize_chr_scalar() and to_chr_scalar() are optimized to check for length-1 character vectors.

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_chr_scalar(
  x,
  ...,
  allow_null = TRUE,
  allow_zero_length = TRUE,
  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)
)

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

Arguments

x	The argument to stabilize.
...	Arguments passed to methods.
allow_null	(length-1 logical) Is NULL an acceptable value?

allow_na (length-1 logical) Are NA values ok?

min_size (length-1 integer) The minimum size of the object. Object size will be tested using vctrs::vec_size().

max_size (length-1 integer) The maximum size of the object. Object size will be tested using vctrs::vec_size().

regex (character, list, or stringr_pattern) One or more optional regular expressions to test against the values of x. This can be a character vector, a list of character vectors, or a pattern object from the {stringr} package (e.g., stringr::fixed("a.b")). The default error message for non-matching values will include the pattern itself (see regex_must_match()). To provide a custom message, supply a named character vector where the value is the regex pattern and the name is the message that should be displayed. To check that a pattern is *not* matched, attach a negate attribute set to TRUE. If a complex regex pattern throws an error, try installing the stringi package.

x_arg (length-1 character) An argument name for x. 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 x to use in error messages. Use this if you remove a special class from x 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 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.

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

`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.

`stabilize_dbl()` can check more details about the argument, but is slower than `to_dbl()`.

`stabilize_dbl_scalar()` and `to_dbl_scalar()` are optimized to check for length-1 double vectors.

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_dbl_scalar(
  x,
  ...,
```

```
  allow_null = TRUE,
  allow_zero_length = TRUE,
  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)
)

## 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 = TRUE,
  allow_zero_length = TRUE,
```

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

Arguments

x	The argument to stabilize.
...	Arguments passed to methods.
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 as.integer() and as.double() use the integer index of the factor.
min_size	(length-1 integer) The minimum size of the object. Object size will be tested using vctrs::vec_size().
max_size	(length-1 integer) The maximum size of the object. Object size will be tested using vctrs::vec_size().
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) An argument name for x. 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 x to use in error messages. Use this if you remove a special class from x 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.

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))
stabilize_dbl_scalar(NULL)
try(stabilize_dbl_scalar(NULL, allow_null = FALSE))
```

`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.

`stabilize_fct()` can check more details about the argument, but is slower than `to_fct()`.

`stabilize_fct_scalar()` and `to_fct_scalar()` are optimized to check for length-1 factors.

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_fct_scalar(
  x,
  ...,
  allow_null = TRUE,
  allow_zero_length = TRUE,
  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)
)

## 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 = TRUE,
  allow_zero_length = TRUE,
  levels = NULL,
  to_na = character(),
  x_arg = caller_arg(x),
  call = caller_env(),
  x_class = object_type(x)
)
```

Arguments

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

max_size (length-1 integer) The maximum size of the object. Object size will be tested using vctrs::vec_size().

levels (character) Expected levels. If NULL (default), the levels will be computed by base::factor().

to_na (character) Values to convert to NA.

x_arg (length-1 character) An argument name for x. 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 x to use in error messages. Use this if you remove a special class from x 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.

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.

stabilize_int() can check more details about the argument, but is slower than to_int().

stabilize_int_scalar() and to_int_scalar() are optimized to check for length-1 integer vectors.

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_int_scalar(
  x,
  ...,
  allow_null = TRUE,
  allow_zero_length = TRUE,
  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)
)

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

Arguments

x	The argument to stabilize.
...	Arguments passed to methods.
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 as.integer() and as.double() use the integer index of the factor.

min_size (length-1 integer) The minimum size of the object. Object size will be tested using vctrs::vec_size().

max_size (length-1 integer) The maximum size of the object. Object size will be tested using vctrs::vec_size().

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) An argument name for x. 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 x to use in error messages. Use this if you remove a special class from x 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 an integer.

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))
stabilize_int_scalar(NULL)
try(stabilize_int_scalar(NULL, allow_null = FALSE))
```

`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.

`stabilize_lgl()` can check more details about the argument, but is slower than `to_lgl()`.

`stabilize_lgl_scalar()` and `to_lgl_scalar()` are optimized to check for length-1 logical vectors.

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_lgl_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)
)

to_lgl(
```

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

Arguments

x	The argument to stabilize.
...	Arguments passed to methods.
allow_null	(length-1 logical) Is NULL an acceptable value?
allow_na	(length-1 logical) Are NA values ok?
min_size	(length-1 integer) The minimum size of the object. Object size will be tested using vctrs::vec_size().
max_size	(length-1 integer) The maximum size of the object. Object size will be tested using vctrs::vec_size().
x_arg	(length-1 character) An argument name for x. 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 x to use in error messages. Use this if you remove a special class from x 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.

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)))
stabilize_lgl_scalar(NULL)
try(stabilize_lgl_scalar(NULL, allow_null = FALSE))
```

Index