Package 'restrictR'

October 5, 2026

Title Composable Runtime Contracts

Version 0.3.0

Description Build reusable validators from small building blocks using the base pipe operator. Define runtime contracts once with restrict() and enforce them anywhere in code. Validators compose naturally, support dependent rules via formulas, and produce clear, path-aware error messages. No domain-specific language, no operator overloading, just idiomatic R.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

URL https://gillescolling.com/restrictR/, https://github.com/gcol33/restrictR

BugReports https://github.com/gcol33/restrictR/issues

Depends R (>= 4.1.0)

Suggests knitr, rmarkdown, svglite, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Gilles Colling [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3070-6066>)

Maintainer Gilles Colling <gilles.colling051@gmail.com>

Repository CRAN

Date/Publication 2026-10-05 12:00:02 UTC

Contents

allow_null . . . 3
as_contract_block . . . 4
as_contract_text . . . 5
expect_invalid . . . 5
expect_valid . . . 6
fail . . . 7
is_valid . . . 8
require_any . . . 9
require_between . . . 10
require_character . . . 11
require_class . . . 11
require_col . . . 12
require_contains . . . 13
require_custom . . . 14
require_df . . . 15
require_dim . . . 16
require_dir_exists . . . 16
require_disjoint . . . 17
require_each . . . 18
require_fields . . . 19
require_file_exists . . . 20
require_finite . . . 21
require_function . . . 21
require_has_cols . . . 22
require_integer . . . 23
require_length . . . 23
require_length_matches . . . 24
require_length_max . . . 25
require_length_min . . . 25
require_levels . . . 26
require_logical . . . 27
require_named . . . 27
require_names . . . 28
require_nchar . . . 29
require_ncol_matches . . . 30
require_ncol_min . . . 30
require_negative . . . 31
require_nonempty . . . 32
require_not_null . . . 32
require_no_na . . . 33
require_nrow_matches . . . 34
require_nrow_min . . . 34
require_numeric . . . 35
require_one_of . . . 36
require_pattern . . . 36
require_positive . . . 37

require_readable 38
require_scalar 39
require_set_equal 39
require_sorted 40
require_unique 41
require_unique_names 41
require_valid 42
require_writable 43
restrict 44
steps 45
validation_errors 46

Index **47**

`allow_null` *Allow NULL*

Description

Marks a validator as accepting `NULL`: a `NULL` value passes without running any step, and every other value is validated by all steps as usual. Use it for optional arguments such as `weights = NULL`.

Usage

```
allow_null(restriction)
```

Arguments

`restriction`	a `restriction` object.

Details

The marker is order-independent: it can sit anywhere in the pipe. A `NULL` value returns before the context check, so context a step needs (e.g. `data` in `require_length_matches(~ nrow(data))`) is not required when the value is `NULL`. A validator that allows `NULL` also accepts a `NULL` (or absent) element when it is lifted with `require_col()`, `require_each()`, `require_fields()` or `require_valid()`.

Value

The modified `restriction` object.

See Also

Other composition: `require_any()`, `require_col()`, `require_each()`, `require_fields()`, `require_valid()`

Examples

```
weights_v <- restrict("weights") |>
  require_numeric(no_na = TRUE) |>
  require_positive() |>
  allow_null()
weights_v(NULL)         # passes
weights_v(c(1, 2))      # passes
is_valid(weights_v, -1) # FALSE
```

---

as_contract_block *Convert a Validator to a Multi-Line Block*

Description

Produces a multi-line text summary suitable for roxygen `@details` documentation. Each step appears on its own line as a bullet point.

Usage

```
as_contract_block(x)
```

Arguments

x	a `restriction` object.

Value

A character(1) string with one step per line.

See Also

Other core: `as_contract_text()`, `fail()`, `is_valid()`, `require_custom()`, `restrict()`, `steps()`, `validation_errors()`

Examples

```
v <- restrict("x") |> require_numeric(no_na = TRUE) |> require_length(1L)
as_contract_block(v)
```

as_contract_text *Convert a Validator to Plain Text*

Description

Produces a single-line text summary suitable for roxygen `@param` documentation. Use with inline R code in roxygen: `` `r as_contract_text(validator)` ``.

Usage

```
as_contract_text(x)
```

Arguments

x	a `restriction` object.

Value

A character(1) string describing the validation contract.

See Also

Other core: `as_contract_block()`, `fail()`, `is_valid()`, `require_custom()`, `restrict()`, `steps()`, `validation_errors()`

Examples

```
v <- restrict("x") |> require_numeric(no_na = TRUE) |> require_length(1L)
as_contract_text(v)
```

expect_invalid *Expect a Value to Fail a Validator*

Description

testthat expectation that the value violates the validator, optionally with a message matching a regular expression.

Usage

```
expect_invalid(validator, value, ..., regexp = NULL)
```

Arguments

validator	a restriction object created by restrict().
value	the value to validate.
...	context arguments passed to the validator (e.g. newdata = df).
regexp	optional regular expression the failure messages must match. All messages are joined with newlines before matching.

Details

Requires the testthat package, which is checked when the expectation runs. A missing context dependency is a usage error and propagates.

Value

value, invisibly.

See Also

expect_valid()

Other testthat expectations: expect_valid()

Examples

```
if (requireNamespace("testthat", quietly = TRUE)) {
  v <- restrict("x") |> require_numeric()
  expect_invalid(v, "a", regexp = "must be numeric")
}
```

---

expect_valid *Expect a Value to Pass a Validator*

Description

testthat expectation that reports the validator's own messages when the value is invalid, so a failing test shows newdata$x2: must be numeric rather than a generic "error thrown".

Usage

```
expect_valid(validator, value, ...)
```

Arguments

validator	a restriction object created by restrict().
value	the value to validate.
...	context arguments passed to the validator (e.g. newdata = df).

Details

Requires the testthat package, which is checked when the expectation runs. Every violation is reported, as with `validation_errors()`.

Value

`value`, invisibly.

See Also

`expect_invalid()`

Other testthat expectations: `expect_invalid()`

Examples

```
if (requireNamespace("testthat", quietly = TRUE)) {
  v <- restrict("x") |> require_numeric()
  expect_valid(v, 1:3)
}
```

`fail` *Format a Validation Error*

Description

Produces a consistently formatted error message and stops execution. Intended for use inside custom validation steps created with `require_custom()`, so they produce the same structured errors as built-in steps.

Usage

```
fail(path, message, found = NULL, at = NULL)
```

Arguments

`path`	the full path (e.g. `"x"` or `"newdata$x2"`).
`message`	the specific failure message.
`found`	optional value to show on a `Found:` line.
`at`	optional integer positions to show on an `At:` line.

Details

Format: `path: message`, with optional `Found:` and `At:` lines.

Value

No return value. Called for its side effect: it signals a classed `restrictR_failure` condition (an error) carrying `path`, `message`, and the optional `found` and `at` details.

See Also

Other core: `as_contract_block()`, `as_contract_text()`, `is_valid()`, `require_custom()`, `restrict()`, `steps()`, `validation_errors()`

Examples

```
# fail() signals an error; wrap in try() to show the formatted message
try(fail("x", "must be positive", found = -3, at = 2L))
```

---

is_valid *Test Whether a Value Satisfies a Validator*

Description

Non-throwing predicate: returns `TRUE` when `value` passes every step of `validator`, `FALSE` otherwise. Use it to branch on validity instead of wrapping a validator call in `tryCatch()`.

Usage

```
is_valid(validator, value, ...)
```

Arguments

`validator`	a `restriction` object created by `restrict()`.
`value`	the value to validate.
`...`	context arguments passed to the validator (e.g. `newdata = df`), the same as a direct validator call.

Details

Like `validation_errors()`, only validation failures are caught; a missing context dependency still raises so a calling mistake is not mistaken for invalid data.

Value

A length-1 logical.

See Also

`validation_errors()` for the failure messages.

Other core: `as_contract_block()`, `as_contract_text()`, `fail()`, `require_custom()`, `restrict()`, `steps()`, `validation_errors()`

Examples

```
v <- restrict("x") |> require_numeric(no_na = TRUE)
is_valid(v, 1:5)        # TRUE
is_valid(v, c(1, NA))   # FALSE
```

`require_any` *Require at Least One of Several Validators*

Description

Passes when the value satisfies at least one of the alternative validators. When none passes, the error lists the first failure of every alternative.

Usage

```
require_any(restriction, ..., .label = NULL)
```

Arguments

`restriction`	a `restriction` object.
`...`	`restriction` objects, the alternatives.
`.label`	optional character(1) description for `print()` and contract text; defaults to the alternatives' own labels.

Details

Each alternative runs in fail-first mode. Context needed by any alternative is required from the caller, like for any other step. Missing context is a usage error, not an alternative failing. In `.on_fail = "all"` mode the step reports a single failure.

Value

The modified `restriction` object.

See Also

Other composition: `allow_null()`, `require_col()`, `require_each()`, `require_fields()`, `require_valid()`

Examples

```
num_or_df <- restrict("x") |>
  require_any(
    restrict("x") |> require_numeric(),
    restrict("x") |> require_df() |> require_has_cols("value")
  )
num_or_df(1:3)
num_or_df(data.frame(value = 1))
```

```
try(num_or_df("a"))
```

`require_between` *Require Value in Range*

Description

Validates that all elements of a value fall within a specified range. The value can be numeric, `Date`, `POSIXct`, `difftime` or an ordered factor.

Usage

```
require_between(
  restriction,
  lower = -Inf,
  upper = Inf,
  exclusive_lower = FALSE,
  exclusive_upper = FALSE
)
```

Arguments

`restriction`	a `restriction` object.
`lower`	lower bound (default `-Inf`, no lower bound).
`upper`	upper bound (default `Inf`, no upper bound).
`exclusive_lower`	logical; if `TRUE`, lower bound is exclusive.
`exclusive_upper`	logical; if `TRUE`, upper bound is exclusive.

Details

The value and the bounds must be the same kind: a `Date` bound against a numeric value fails with a type error. Ordered-factor bounds are single values of a factor with the same levels as the validated value. Other input fails with a type error. `NA` elements are skipped; chain `require_no_na()` to reject them.

Value

The modified `restriction` object.

See Also

Other value checks: `require_contains()`, `require_disjoint()`, `require_levels()`, `require_negative()`, `require_one_of()`, `require_positive()`, `require_set_equal()`, `require_unique()`

Examples

```
period <- restrict("start") |>
  require_between(as.Date("2020-01-01"), as.Date("2020-12-31"))
period(as.Date("2020-06-15"))
try(period(as.Date("2021-01-01")))
```

---

require_character *Require Character Type*

Description

Validates that the value is character. Optionally checks for NA values.

Usage

```
require_character(restriction, no_na = FALSE)
```

Arguments

restriction	a restriction object.
no_na	logical; if TRUE, rejects NA values.

Value

The modified restriction object.

See Also

Other type checks: require_class(), require_df(), require_integer(), require_logical(), require_numeric()

---

require_class *Require a Specific Class*

Description

Validates that the value belongs to a given class. One verb covers the types without a dedicated check, including factor, Date, POSIXct, list, matrix, environment, and fitted-model objects such as lm.

Usage

```
require_class(restriction, class, exact = FALSE)
```

Arguments

restriction	a restriction object.
class	character(1) class name to require.
exact	logical; if TRUE, requires class(value)[1] to equal class exactly. If FALSE (default), tests inheritance with inherits(), so a subclass passes.

Details

A matrix is require_class("matrix") (inherits() is TRUE for matrices since R 4.0) and an environment is require_class("environment"); neither needs a dedicated step. Use require_dim() for the shape.

Value

The modified restriction object.

See Also

Other type checks: require_character(), require_df(), require_integer(), require_logical(), require_numeric()

Examples

```
restrict("d") |> require_class("Date")
restrict("f") |> require_class("factor")
restrict("model") |> require_class("lm")
restrict("m") |> require_class("matrix") |> require_dim(c(NA, 3))
```

---

require_col *Validate a Data Frame Column with a Validator*

Description

Lifts any validator onto one column of a data.frame. Errors keep the column path (e.g. newdata$age: must be in [0, 120]) so one set of steps serves both a standalone argument and a column.

Usage

```
require_col(restriction, col, validator)
```

Arguments

restriction	a restriction object.
col	character(1) column name.
validator	a restriction object applied to the column.

Details

The value must be a data.frame containing `col`, unless `validator` allows `NULL` (see `allow_null()`), in which case a missing column passes. Context passed to the outer validator is visible to formula steps in `validator`. With `.on_fail = "all"` each inner failure is reported on its own.

Value

The modified `restriction` object.

See Also

Other composition: `allow_null()`, `require_any()`, `require_each()`, `require_fields()`, `require_valid()`

Examples

```
age <- restrict("age") |>
  require_integer(no_na = TRUE) |>
  require_between(0, 120)

newdata_v <- restrict("newdata") |>
  require_df() |>
  require_col("age", age) |>
  require_col("sex", restrict("sex") |> require_one_of(c("f", "m")))

newdata_v(data.frame(age = 30L, sex = "f"))
validation_errors(newdata_v, data.frame(age = 130L, sex = "x"))
```

`require_contains` *Require All of a Set of Values*

Description

Validates that the value contains every element of `values`. Extra elements are allowed.

Usage

```
require_contains(restriction, values)
```

Arguments

`restriction`	a `restriction` object.
`values`	vector of values that must be present.

Details

`NA` elements of the validated value are ignored. A factor is compared by its labels. For the opposite direction (every element must be one of `values`) use `require_one_of()`.

Value

The modified `restriction` object.

See Also

Other value checks: `require_between()`, `require_disjoint()`, `require_levels()`, `require_negative()`, `require_one_of()`, `require_positive()`, `require_set_equal()`, `require_unique()`

`require_custom` *Create a Custom Validation Step*

Description

Allows advanced users to define their own validation step without growing the package's built-in API surface. The step function receives `(value, name, ctx)` and should call `fail()` on validation failure.

Usage

```
require_custom(restriction, label, fn, deps = character(0L))
```

Arguments

`restriction`	a `restriction` object.
`label`	character(1) human-readable description for printing.
`fn`	a function with signature `function(value, name, ctx)` that calls `fail()` on validation failure. It must be callable with three positional arguments (see `require_function()`).
`deps`	character vector of context names this step requires (default: none).

Value

A new `restriction` object with the custom step appended.

See Also

Other core: `as_contract_block()`, `as_contract_text()`, `fail()`, `is_valid()`, `restrict()`, `steps()`, `validation_errors()`

Examples

```
# Custom step: require all values to be unique
require_unique_id <- restrict("id") |>
  require_custom(
    label = "must contain unique values",
    fn = function(value, name, ctx) {
      dupes <- which(duplicated(value))
      if (length(dupes) > 0L) {
        fail(name, "contains duplicates", at = dupes)
      }
    }
  )
```

`require_df` *Require a Data Frame*

Description

Validates that the value is a `data.frame`.

Usage

```
require_df(restriction)
```

Arguments

`restriction`	a `restriction` object.

Value

The modified `restriction` object.

See Also

Other type checks: `require_character()`, `require_class()`, `require_integer()`, `require_logical()`, `require_numeric()`

require_dim *Require Exact Dimensions*

Description

Validates `dim(value)` of a data.frame, matrix or array. `NA` entries in `dims` match any extent.

Usage

```
require_dim(restriction, dims)
```

Arguments

`restriction`	a `restriction` object.
`dims`	numeric vector of required extents, one per dimension; `NA` leaves that dimension unchecked.

Value

The modified `restriction` object.

See Also

Other structure checks: `require_has_cols()`, `require_length()`, `require_length_matches()`, `require_length_max()`, `require_length_min()`, `require_named()`, `require_names()`, `require_ncol_matches()`, `require_ncol_min()`, `require_nrow_matches()`, `require_nrow_min()`, `require_scalar()`, `require_sorted()`, `require_unique_names()`

Examples

```
design <- restrict("X") |> require_class("matrix") |> require_dim(c(NA, 3))
design(matrix(0, 10, 3))
try(design(matrix(0, 10, 2)))
```

require_dir_exists *Require Existing Directories*

Description

Validates that every path of a character value exists and is a directory.

Usage

```
require_dir_exists(restriction)
```

Arguments

`restriction`	a `restriction` object.

Details

Non-character input fails with a type error. `NA` paths are skipped; chain `require_no_na()` to reject them.

Value

The modified `restriction` object.

See Also

Other file checks: `require_file_exists()`, `require_readable()`, `require_writable()`

`require_disjoint` *Require Values Disjoint From Context*

Description

Validates that no element of the value occurs in the vector a formula evaluates to, for example that train and test ids do not overlap. The formula is evaluated using only explicitly passed context arguments, plus `.value` and `.name`.

Usage

```
require_disjoint(restriction, formula)
```

Arguments

`restriction`	a `restriction` object.
`formula`	a one-sided formula (e.g. `~ test_ids`).

Details

`NA` elements are ignored. A factor is compared by its labels.

Value

The modified `restriction` object.

See Also

Other value checks: `require_between()`, `require_contains()`, `require_levels()`, `require_negative()`, `require_one_of()`, `require_positive()`, `require_set_equal()`, `require_unique()`

Examples

```
train_ids <- restrict("train_ids") |> require_disjoint(~ test_ids)
train_ids(1:3, test_ids = 4:6)
try(train_ids(1:3, test_ids = 3:6))
```

`require_each` *Validate Every Element of a List*

Description

Applies a validator to each element of a list: list arguments, list-columns, collected `...`, configuration lists. Errors carry the element path, `layers[[2]]` for unnamed elements and `layers$key` for named ones.

Usage

```
require_each(restriction, validator)
```

Arguments

`restriction`	a `restriction` object.
`validator`	a `restriction` object applied to every element.

Details

The value must be a list. An empty list passes. In fail-first mode the first invalid element stops the check; with `.on_fail = "all"` every failing element is reported.

Value

The modified `restriction` object.

See Also

Other composition: `allow_null()`, `require_any()`, `require_col()`, `require_fields()`, `require_valid()`

Examples

```
layer <- restrict("layer") |> require_class("data.frame")
layers_v <- restrict("layers") |> require_class("list") |> require_each(layer)
layers_v(list(data.frame(a = 1), data.frame(b = 2)))
try(layers_v(list(data.frame(a = 1), "oops")))
```

`require_fields` *Validate Named Fields of a List*

Description

Applies a different validator to each named element of a list, for heterogeneous configuration or argument lists. Errors carry the field path, e.g. `opts$alpha`.

Usage

```
require_fields(restriction, ..., .required = TRUE)
```

Arguments

`restriction`	a `restriction` object.
`...`	named `restriction` objects, one per field.
`.required`	logical; if `TRUE` (default) a missing field is a failure. If `FALSE`, a missing field passes. A field whose validator allows `NULL` (see `allow_null()`) may always be missing.

Value

The modified `restriction` object.

See Also

Other composition: `allow_null()`, `require_any()`, `require_col()`, `require_each()`, `require_valid()`

Examples

```
opts_v <- restrict("opts") |>
  require_class("list") |>
  require_fields(
    alpha = restrict("alpha") |> require_numeric() |> require_between(0, 1),
    label = restrict("label") |> require_character()
  )
opts_v(list(alpha = 0.05, label = "a"))
validation_errors(opts_v, list(alpha = 2))
```

`require_file_exists` *Require Existing Files*

Description

Validates that every path of a character value exists and is a file. `Found:` shows the normalized path of the first offender, so a relative path resolved against the wrong working directory is visible.

Usage

```
require_file_exists(restriction, extension = NULL)
```

Arguments

`restriction`	a `restriction` object.
`extension`	optional character vector of accepted extensions, with or without the leading dot, compared case-insensitively. Adds a second step that checks the extension.

Details

Non-character input fails with a type error. `NA` paths are skipped; chain `require_no_na()` to reject them.

Value

The modified `restriction` object.

See Also

Other file checks: `require_dir_exists()`, `require_readable()`, `require_writable()`

Examples

```
csv_in <- restrict("path") |> require_file_exists(extension = "csv")
tmp <- tempfile(fileext = ".csv")
invisible(file.create(tmp))
csv_in(tmp)
try(csv_in(file.path(tempdir(), "missing.csv")))
```

require_finite *Require Finite Values*

Description

Validates that a numeric value contains no `Inf`, `-Inf`, or `NaN` values. Does not check for `NA` (use `require_no_na()` for that).

Usage

```
require_finite(restriction)
```

Arguments

`restriction`	a `restriction` object.

Value

The modified `restriction` object.

See Also

Other missingness checks: `require_no_na()`, `require_not_null()`

require_function *Require a Function*

Description

Validates that the value is a function, optionally with named arguments or a call signature. Meant for callback arguments.

Usage

```
require_function(restriction, args = NULL, nargs = NULL)
```

Arguments

`restriction`	a `restriction` object.
`args`	optional character vector of argument names the function must declare.
`nargs`	optional whole number: the function must be callable with exactly this many positional arguments, so it declares at least that many (or `...`) and requires no more.

Details

`args` looks at the declared formals only: a function declaring `...` does not satisfy a named argument.

Value

The modified `restriction` object.

Examples

```
callback <- restrict("fn") |> require_function(nargs = 2L)
callback(function(x, y) x + y)
try(callback(function(x) x))
```

---

require_has_cols *Require Specific Columns*

Description

Validates that a data.frame contains all specified columns. Equivalent to `require_names(cols, mode = "superset")` with column wording in the message.

Usage

```
require_has_cols(restriction, cols)
```

Arguments

`restriction`	a `restriction` object.
`cols`	character vector of required column names.

Value

The modified `restriction` object.

See Also

Other structure checks: `require_dim()`, `require_length()`, `require_length_matches()`, `require_length_max()`, `require_length_min()`, `require_named()`, `require_names()`, `require_ncol_matches()`, `require_ncol_min()`, `require_nrow_matches()`, `require_nrow_min()`, `require_scalar()`, `require_sorted()`, `require_unique_names()`

require_integer *Require Integer Values*

Description

Validates that the value contains whole numbers. By default accepts both `integer` and `numeric` types as long as all values are whole (`x == floor(x)`). Set `strict = TRUE` to require the R `integer` type.

Usage

```
require_integer(restriction, no_na = FALSE, strict = FALSE)
```

Arguments

`restriction`	a `restriction` object.
`no_na`	logical; if `TRUE`, rejects NA values.
`strict`	logical; if `TRUE`, requires R `integer` type. If `FALSE` (default), accepts any numeric value that is a whole number.

Details

`Inf` and `-Inf` are not whole numbers and are rejected in both modes; `NaN` counts as `NA`.

Value

The modified `restriction` object.

See Also

Other type checks: `require_character()`, `require_class()`, `require_df()`, `require_logical()`, `require_numeric()`

require_length *Require Specific Length*

Description

Validates that the value has exact length `n`.

Usage

```
require_length(restriction, n)
```

Arguments

restriction	a restriction object.
n	integer(1) required length.

Value

The modified restriction object.

See Also

Other structure checks: require_dim(), require_has_cols(), require_length_matches(), require_length_max(), require_length_min(), require_named(), require_names(), require_ncol_matches(), require_ncol_min(), require_nrow_matches(), require_nrow_min(), require_scalar(), require_sorted(), require_unique_names()

---

require_length_matches

Require Length Matching an Expression

Description

Validates that length(value) equals the result of evaluating a formula. The formula is evaluated using only explicitly passed context arguments, plus .value (the validated value) and .name (the restriction name).

Usage

```
require_length_matches(restriction, formula)
```

Arguments

restriction	a restriction object.
formula	a one-sided formula (e.g. ~ nrow(newdata)).

Value

The modified restriction object.

See Also

Other structure checks: require_dim(), require_has_cols(), require_length(), require_length_max(), require_length_min(), require_named(), require_names(), require_ncol_matches(), require_ncol_min(), require_nrow_matches(), require_nrow_min(), require_scalar(), require_sorted(), require_unique_names()

require_length_max *Require Maximum Length*

Description

Validates that the value has at most length `n`.

Usage

```
require_length_max(restriction, n)
```

Arguments

`restriction`	a `restriction` object.
`n`	integer(1) maximum length.

Value

The modified `restriction` object.

See Also

Other structure checks: `require_dim()`, `require_has_cols()`, `require_length()`, `require_length_matches()`, `require_length_min()`, `require_named()`, `require_names()`, `require_ncol_matches()`, `require_ncol_min()`, `require_nrow_matches()`, `require_nrow_min()`, `require_scalar()`, `require_sorted()`, `require_unique_names()`

require_length_min *Require Minimum Length*

Description

Validates that the value has at least length `n`.

Usage

```
require_length_min(restriction, n)
```

Arguments

`restriction`	a `restriction` object.
`n`	integer(1) minimum length.

Value

The modified `restriction` object.

See Also

Other structure checks: require_dim(), require_has_cols(), require_length(), require_length_matches(), require_length_max(), require_named(), require_names(), require_ncol_matches(), require_ncol_min(), require_nrow_matches(), require_nrow_min(), require_scalar(), require_sorted(), require_unique_names()

`require_levels` *Require Factor Levels*

Description

Compares `levels(value)` of a factor with a reference. The classic `predict(newdata)` failure, a factor with a level the model has not seen, is `require_levels(levels(train$f), mode = "subset")`.

Usage

```
require_levels(
  restriction,
  levels,
  mode = c("identical", "subset", "superset")
)
```

Arguments

`restriction`	a `restriction` object.
`levels`	character vector of reference levels.
`mode`	how the observed levels relate to `levels`:

- `"identical"`: the same levels in the same order;
- `"subset"`: every observed level is in `levels`;
- `"superset"`: every level in `levels` is present.

Details

The levels of the factor are compared, whether or not a level occurs in the data. Non-factor input fails with a type error.

Value

The modified `restriction` object.

See Also

Other value checks: require_between(), require_contains(), require_disjoint(), require_negative(), require_one_of(), require_positive(), require_set_equal(), require_unique()

Examples

```
f_v <- restrict("f") |> require_levels(c("a", "b"), mode = "subset")
f_v(factor(c("a", "b")))
try(f_v(factor(c("a", "z"))))
```

---

require_logical *Require Logical Type*

Description

Validates that the value is logical. Optionally checks for NA values.

Usage

```
require_logical(restriction, no_na = FALSE)
```

Arguments

restriction	a restriction object.
no_na	logical; if TRUE, rejects NA values.

Value

The modified restriction object.

See Also

Other type checks: require_character(), require_class(), require_df(), require_integer(), require_numeric()

---

require_named *Require Named Value*

Description

Validates that the value is fully named: every element has a non-empty, non-NA name. A partially-named value (e.g. c(a = 1, 2)) fails and the positions of the unnamed elements are reported.

Usage

```
require_named(restriction)
```

Arguments

restriction	a restriction object.

Value

The modified `restriction` object.

See Also

Other structure checks: `require_dim()`, `require_has_cols()`, `require_length()`, `require_length_matches()`, `require_length_max()`, `require_length_min()`, `require_names()`, `require_ncol_matches()`, `require_ncol_min()`, `require_nrow_matches()`, `require_nrow_min()`, `require_scalar()`, `require_sorted()`, `require_unique_names()`

`require_names` *Require Names*

Description

Compares `names(value)` with a reference. One verb covers "exactly these names in this order", "only allowed names" and "at least these names".

Usage

```
require_names(
  restriction,
  names,
  mode = c("identical", "subset", "superset", "permutation")
)
```

Arguments

`restriction`	a `restriction` object.
`names`	character vector of reference names.
`mode`	how the observed names relate to `names`:

- `"identical"`: the same names in the same order;
- `"subset"`: every observed name is in `names` (no unexpected names);
- `"superset"`: every name in `names` is present (nothing missing);
- `"permutation"`: the same names in any order, each occurring as often.

Details

A value without names has no names: it fails every mode except `"subset"`. The failure lists the missing and the unexpected names.

Value

The modified `restriction` object.

See Also

Other structure checks: require_dim(), require_has_cols(), require_length(), require_length_matches(), require_length_max(), require_length_min(), require_named(), require_ncol_matches(), require_ncol_min(), require_nrow_matches(), require_nrow_min(), require_scalar(), require_sorted(), require_unique_names()

Examples

```
cfg <- restrict("cfg") |> require_names(c("alpha", "beta"), mode = "subset")
cfg(list(alpha = 1))
try(cfg(list(alpha = 1, gamma = 2)))
```

---

require_nchar *Require String Length*

Description

Validates the number of characters of every non-NA element of a character value.

Usage

```
require_nchar(restriction, min = 0, max = Inf)
```

Arguments

restriction	a restriction object.
min	minimum number of characters (default 0).
max	maximum number of characters (default Inf).

Details

Length is counted in characters, not bytes. Non-character input fails with a type error. NA elements are skipped; chain require_no_na() to reject them.

Value

The modified restriction object.

See Also

Other character checks: require_nonempty(), require_pattern()

Examples

```
initials <- restrict("initials") |> require_nchar(min = 2, max = 3)
initials(c("AB", "ABC"))
try(initials("A"))
```

require_ncol_matches *Require Column Count Matching an Expression*

Description

Validates that `ncol(value)` equals the result of evaluating a formula. The formula is evaluated using only explicitly passed context arguments, plus `.value` (the validated value) and `.name` (the restriction name).

Usage

```
require_ncol_matches(restriction, formula)
```

Arguments

`restriction`	a `restriction` object.
`formula`	a one-sided formula (e.g. `~ ncol(reference)`).

Value

The modified `restriction` object.

See Also

Other structure checks: `require_dim()`, `require_has_cols()`, `require_length()`, `require_length_matches()`, `require_length_max()`, `require_length_min()`, `require_named()`, `require_names()`, `require_ncol_min()`, `require_nrow_matches()`, `require_nrow_min()`, `require_scalar()`, `require_sorted()`, `require_unique_names()`

require_ncol_min *Require Minimum Number of Columns*

Description

Validates that a data.frame or matrix has at least `n` columns.

Usage

```
require_ncol_min(restriction, n)
```

Arguments

`restriction`	a `restriction` object.
`n`	integer(1) minimum column count.

Value

The modified `restriction` object.

See Also

Other structure checks: `require_dim()`, `require_has_cols()`, `require_length()`, `require_length_matches()`, `require_length_max()`, `require_length_min()`, `require_named()`, `require_names()`, `require_ncol_matches()`, `require_nrow_matches()`, `require_nrow_min()`, `require_scalar()`, `require_sorted()`, `require_unique_names()`

`require_negative` *Require Negative Values*

Description

Validates that all elements are negative. By default uses `<= 0` (non-positive); set `strict = TRUE` for `< 0`.

Usage

```
require_negative(restriction, strict = FALSE)
```

Arguments

`restriction`	a `restriction` object.
`strict`	logical; if `TRUE`, requires `< 0`. If `FALSE` (default), requires `<= 0`.

Details

Non-numeric input fails with a type error. `NA` elements are skipped; chain `require_no_na()` to reject them.

Value

The modified `restriction` object.

See Also

Other value checks: `require_between()`, `require_contains()`, `require_disjoint()`, `require_levels()`, `require_one_of()`, `require_positive()`, `require_set_equal()`, `require_unique()`

require_nonempty *Require Non-Blank Strings*

Description

Validates that no non-NA element of a character value is empty or made of whitespace only.

Usage

```
require_nonempty(restriction)
```

Arguments

restriction	a restriction object.

Details

Non-character input fails with a type error. NA elements are skipped; chain require_no_na() to reject them.

Value

The modified restriction object.

See Also

Other character checks: require_nchar(), require_pattern()

Examples

```
label <- restrict("label") |> require_nonempty()
label(c("a", "b"))
try(label(c("a", " ")))
```

require_not_null *Require Non-NULL Value*

Description

Validates that the value is not NULL. Place this step first in the pipeline when NULL is a possible input.

Usage

```
require_not_null(restriction)
```

Arguments

restriction	a restriction object.

Value

The modified restriction object.

See Also

Other missingness checks: require_finite(), require_no_na()

---

require_no_na *Require No NA Values*

Description

Validates that the value contains no NA values. Works on any atomic type.

Usage

```
require_no_na(restriction)
```

Arguments

restriction	a restriction object.

Value

The modified restriction object.

See Also

Other missingness checks: require_finite(), require_not_null()

require_nrow_matches *Require Row Count Matching an Expression*

Description

Validates that `nrow(value)` equals the result of evaluating a formula. The formula is evaluated using only explicitly passed context arguments, plus `.value` (the validated value) and `.name` (the restriction name).

Usage

```
require_nrow_matches(restriction, formula)
```

Arguments

`restriction`	a `restriction` object.
`formula`	a one-sided formula (e.g. `~ nrow(reference)`).

Value

The modified `restriction` object.

See Also

Other structure checks: `require_dim()`, `require_has_cols()`, `require_length()`, `require_length_matches()`, `require_length_max()`, `require_length_min()`, `require_named()`, `require_names()`, `require_ncol_matches()`, `require_ncol_min()`, `require_nrow_min()`, `require_scalar()`, `require_sorted()`, `require_unique_names()`

require_nrow_min *Require Minimum Number of Rows*

Description

Validates that a data.frame or matrix has at least `n` rows.

Usage

```
require_nrow_min(restriction, n)
```

Arguments

`restriction`	a `restriction` object.
`n`	integer(1) minimum row count.

Value

The modified `restriction` object.

See Also

Other structure checks: `require_dim()`, `require_has_cols()`, `require_length()`, `require_length_matches()`, `require_length_max()`, `require_length_min()`, `require_named()`, `require_names()`, `require_ncol_matches()`, `require_ncol_min()`, `require_nrow_matches()`, `require_scalar()`, `require_sorted()`, `require_unique_names()`

---

require_numeric *Require Numeric Type*

Description

Validates that the value is numeric. Optionally checks for NA and non-finite values.

Usage

```
require_numeric(restriction, no_na = FALSE, finite = FALSE)
```

Arguments

`restriction`	a `restriction` object.
`no_na`	logical; if `TRUE`, rejects NA values.
`finite`	logical; if `TRUE`, rejects `Inf`/`-Inf`/`NaN`.

Value

The modified `restriction` object.

See Also

Other type checks: `require_character()`, `require_class()`, `require_df()`, `require_integer()`, `require_logical()`

require_one_of *Require Value from a Set*

Description

Validates that all elements of the value are among the allowed values. For a vector this is the subset test: every element must be one of `values`.

Usage

```
require_one_of(restriction, values)
```

Arguments

`restriction`	a `restriction` object.
`values`	vector of allowed values.

Details

`NA` elements are skipped; chain `require_no_na()` to reject them. Use `require_contains()` for the reverse direction (the value must hold all of `values`) and `require_set_equal()` for both.

Value

The modified `restriction` object.

See Also

Other value checks: `require_between()`, `require_contains()`, `require_disjoint()`, `require_levels()`, `require_negative()`, `require_positive()`, `require_set_equal()`, `require_unique()`

require_pattern *Require Strings Matching a Pattern*

Description

Validates that every non-`NA` element of a character value matches a regular expression. `Found:` shows the first element that does not match and `At:` lists every offender.

Usage

```
require_pattern(restriction, regex, fixed = FALSE, ignore_case = FALSE)
```

Arguments

restriction	a restriction object.
regex	character(1) regular expression (extended syntax, as in grepl()).
fixed	logical; if TRUE, regex is a literal substring.
ignore_case	logical; if TRUE, matching ignores case. Not available with fixed = TRUE.

Details

A match anywhere in the string counts; anchor the pattern with ^ and $ to match the whole string. Non-character input fails with a type error. NA elements are skipped; chain require_no_na() to reject them.

Value

The modified restriction object.

See Also

Other character checks: require_nchar(), require_nonempty()

Examples

```
code <- restrict("code") |> require_character() |> require_pattern("^[A-Z]{3}$")
code(c("ABC", "XYZ"))
try(code(c("ABC", "xy")))
```

---

require_positive *Require Positive Values*

Description

Validates that all elements are positive. By default uses >= 0 (non-negative); set strict = TRUE for > 0.

Usage

```
require_positive(restriction, strict = FALSE)
```

Arguments

restriction	a restriction object.
strict	logical; if TRUE, requires > 0. If FALSE (default), requires >= 0.

Details

Non-numeric input fails with a type error. NA elements are skipped; chain require_no_na() to reject them.

Value

The modified `restriction` object.

See Also

Other value checks: `require_between()`, `require_contains()`, `require_disjoint()`, `require_levels()`, `require_negative()`, `require_one_of()`, `require_set_equal()`, `require_unique()`

`require_readable` *Require Readable Paths*

Description

Validates that every path of a character value exists and can be read, judged by `file.access()`.

Usage

```
require_readable(restriction)
```

Arguments

`restriction`	a `restriction` object.

Details

Non-character input fails with a type error. `NA` paths are skipped; chain `require_no_na()` to reject them.

Value

The modified `restriction` object.

See Also

Other file checks: `require_dir_exists()`, `require_file_exists()`, `require_writable()`

require_scalar *Require Scalar Value*

Description

Validates that the value has length 1. Rejects `NULL`, zero-length vectors, and vectors with more than one element.

Usage

```
require_scalar(restriction)
```

Arguments

`restriction`	a `restriction` object.

Value

The modified `restriction` object.

See Also

Other structure checks: `require_dim()`, `require_has_cols()`, `require_length()`, `require_length_matches()`, `require_length_max()`, `require_length_min()`, `require_named()`, `require_names()`, `require_ncol_matches()`, `require_ncol_min()`, `require_nrow_matches()`, `require_nrow_min()`, `require_sorted()`, `require_unique_names()`

require_set_equal *Require the Same Set of Values*

Description

Validates that the value and `values` hold the same distinct elements. Order and duplicates are ignored.

Usage

```
require_set_equal(restriction, values)
```

Arguments

`restriction`	a `restriction` object.
`values`	vector of values that must be present.

Value

The modified `restriction` object.

See Also

Other value checks: require_between(), require_contains(), require_disjoint(), require_levels(), require_negative(), require_one_of(), require_positive(), require_unique()

`require_sorted` *Require Sorted Values*

Description

Validates that the elements are in order and reports the position of the first element that breaks it.

Usage

```
require_sorted(restriction, decreasing = FALSE, strict = FALSE)
```

Arguments

`restriction`	a `restriction` object.
`decreasing`	logical; if `TRUE`, requires decreasing order.
`strict`	logical; if `TRUE`, equal neighbours are a violation.

Details

Works on numeric, character, logical, `Date`, `POSIXct`, `difftime` and ordered-factor values; other input fails with a type error. Character values compare in the collation order of the session locale. `NA` elements are skipped; chain require_no_na() to reject them.

Value

The modified `restriction` object.

See Also

Other structure checks: require_dim(), require_has_cols(), require_length(), require_length_matches(), require_length_max(), require_length_min(), require_named(), require_names(), require_ncol_matches(), require_ncol_min(), require_nrow_matches(), require_nrow_min(), require_scalar(), require_unique_names()

Examples

```
ts_v <- restrict("time") |> require_sorted(strict = TRUE)
ts_v(c(1, 2, 5))
try(ts_v(c(1, 3, 2)))
```

require_unique *Require Unique Values*

Description

Validates that the value contains no duplicates. Reports the positions of duplicated elements.

Usage

```
require_unique(restriction)
```

Arguments

restriction	a restriction object.

Value

The modified restriction object.

See Also

Other value checks: require_between(), require_contains(), require_disjoint(), require_levels(), require_negative(), require_one_of(), require_positive(), require_set_equal()

require_unique_names *Require Unique Names*

Description

Validates that no name occurs twice. Unnamed elements are ignored; use require_named() to reject them.

Usage

```
require_unique_names(restriction)
```

Arguments

restriction	a restriction object.

Value

The modified restriction object.

See Also

Other structure checks: require_dim(), require_has_cols(), require_length(), require_length_matches(), require_length_max(), require_length_min(), require_named(), require_names(), require_ncol_matches(), require_ncol_min(), require_nrow_matches(), require_nrow_min(), require_scalar(), require_sorted()

---

require_valid *Include Another Validator's Steps*

Description

Splices the steps of `validator` into `restriction`, so two independently defined validators combine into one. Splicing happens when the validator is built: `print()` shows one flat list of steps and there is no nesting at run time.

Usage

```
require_valid(restriction, validator)
```

Arguments

`restriction`	a `restriction` object.
`validator`	a `restriction` object whose steps are appended.

Details

If `validator` allows `NULL` (see allow_null()), its steps are skipped for a `NULL` value and the combined validator does not itself become `NULL`-tolerant.

Value

The modified `restriction` object.

See Also

Other composition: allow_null(), require_any(), require_col(), require_each(), require_fields()

Examples

```
id_v <- restrict("id") |> require_integer(no_na = TRUE)
pos_v <- restrict("x") |> require_positive(strict = TRUE)
restrict("id") |> require_valid(id_v) |> require_valid(pos_v)
```

`require_writable` *Require Writable Paths*

Description

Validates that every path of a character value can be written: an existing file or directory must be writable, and a path that does not exist yet needs a writable parent directory. Use it for output files and directories.

Usage

```
require_writable(restriction)
```

Arguments

`restriction`	a `restriction` object.

Details

Writability is judged by `file.access()`. On Windows it reflects the read-only attribute and not access control lists, so a directory that is denied by permissions can still pass. The check cannot tell whether the disk has room.

Value

The modified `restriction` object.

See Also

Other file checks: `require_dir_exists()`, `require_file_exists()`, `require_readable()`

Examples

```
out_dir <- restrict("out") |> require_writable()
out_dir(tempdir())
out_dir(file.path(tempdir(), "result.csv"))
```

`restrict` *Create a Composable Validator*

Description

Creates a callable validation object that accumulates checks via the base pipe operator `|>`. The resulting object behaves like a function: call it with a value to validate.

Usage

```
restrict(name)
```

Arguments

`name`	character(1) name used in error messages (e.g. `"newdata"`).

Value

A `restriction` object (callable function) with no validation steps.

Calling convention

Validators accept `value` as the first argument, plus context via named arguments in `...` or as a named list in `.ctx`:

```
require_pred(out, newdata = df)
require_pred(out, .ctx = list(newdata = df))
```

Named arguments in `...` take precedence over `.ctx` entries with the same name. If a step declares dependencies (e.g. `require_length_matches(~ nrow(newdata))`), the validator checks that all required context is present before running any steps and errors early if not.

By default a validator is fail-fast: the first failing step stops with its error. Pass `.on_fail = "all"` to run every step and report all violations in one aggregated error:

```
require_pred(out, .on_fail = "all")
```

In `"all"` mode a type or structure guard (wrong type, not a data.frame, missing column) is reported once per path: later steps that fail the same guard on the same path are not repeated. Independent value failures are all reported.

For a non-throwing result, use `is_valid()` or `validation_errors()`.

See Also

Other core: `as_contract_block()`, `as_contract_text()`, `fail()`, `is_valid()`, `require_custom()`, `steps()`, `validation_errors()`

Examples

```
# Define a validator
require_positive <- restrict("x") |>
  require_numeric(no_na = TRUE) |>
  require_between(lower = 0, exclusive_lower = TRUE)

# Use it
require_positive(5)   # passes silently

# Compose with pipe
require_score <- restrict("score") |>
  require_numeric() |>
  require_length(1L) |>
  require_between(lower = 0, upper = 100)
```

---

steps *List the Steps of a Validator*

Description

Returns the steps of a validator as a data.frame, one row per step in the order they run, for introspection and tooling. It shows the same steps as `print()`.

Usage

```
steps(x)
```

Arguments

x	a `restriction` object.

Value

A data.frame with the columns `step` (position), `label` (the description shown by `print()`), `deps` (list column: context names the step needs) and `fields` (list column: the parameters the step was built with, `NULL` when it has none).

See Also

Other core: `as_contract_block()`, `as_contract_text()`, `fail()`, `is_valid()`, `require_custom()`, `restrict()`, `validation_errors()`

Examples

```
v <- restrict("x") |> require_numeric(no_na = TRUE) |> require_between(0, 1)
steps(v)
steps(v)$fields[[2]]
```

validation_errors *Collect Validation Errors Without Throwing*

Description

Runs a validator against a value and returns the validation messages instead of raising an error. Every step is checked (as with `.on_fail = "all"`), so all violations are reported in one pass.

Usage

```
validation_errors(validator, value, ...)
```

Arguments

`validator`	a `restriction` object created by `restrict()`.
`value`	the value to validate.
`...`	context arguments passed to the validator (e.g. `newdata = df`), the same as a direct validator call.

Details

Only validation failures are caught. A usage error, such as a missing context dependency declared by a formula step, still propagates so the calling mistake is not silently reported as invalid data.

Value

A character vector of failure messages, or `character(0)` if the value satisfies every step. Each element is the full path-aware message for one failing step.

See Also

`is_valid()` for a logical predicate.

Other core: `as_contract_block()`, `as_contract_text()`, `fail()`, `is_valid()`, `require_custom()`, `restrict()`, `steps()`

Examples

```
v <- restrict("x") |> require_numeric() |> require_positive()
validation_errors(v, 5)       # character(0)
validation_errors(v, c(-1))   # one message
```

Index

∗ **character checks**
  require_nchar, 29
  require_nonempty, 32
  require_pattern, 36
∗ **composition**
  allow_null, 3
  require_any, 9
  require_col, 12
  require_each, 18
  require_fields, 19
  require_valid, 42
∗ **core**
  as_contract_block, 4
  as_contract_text, 5
  fail, 7
  is_valid, 8
  require_custom, 14
  restrict, 44
  steps, 45
  validation_errors, 46
∗ **file checks**
  require_dir_exists, 16
  require_file_exists, 20
  require_readable, 38
  require_writable, 43
∗ **function checks**
  require_function, 21
∗ **missingness checks**
  require_finite, 21
  require_no_na, 33
  require_not_null, 32
∗ **structure checks**
  require_dim, 16
  require_has_cols, 22
  require_length, 23
  require_length_matches, 24
  require_length_max, 25
  require_length_min, 25
  require_named, 27
  require_names, 28
  require_ncol_matches, 30
  require_ncol_min, 30
  require_nrow_matches, 34
  require_nrow_min, 34
  require_scalar, 39
  require_sorted, 40
  require_unique_names, 41
∗ **testthat expectations**
  expect_invalid, 5
  expect_valid, 6
∗ **type checks**
  require_character, 11
  require_class, 11
  require_df, 15
  require_integer, 23
  require_logical, 27
  require_numeric, 35
∗ **value checks**
  require_between, 10
  require_contains, 13
  require_disjoint, 17
  require_levels, 26
  require_negative, 31
  require_one_of, 36
  require_positive, 37
  require_set_equal, 39
  require_unique, 41

allow_null, 3
allow_null(), *9*, *13*, *18, 19*, *42*
as_contract_block, 4
as_contract_block(), *5*, *8*, *14*, *44–46*
as_contract_text, 5
as_contract_text(), *4*, *8*, *14*, *44–46*

expect_invalid, 5
expect_invalid(), *7*
expect_valid, 6
expect_valid(), *6*

fail, 7
fail(), *4, 5*, *8*, *14*, *44–46*
file.access(), *38*, *43*

grepl(), *37*

inherits(), *12*
is_valid, 8
is_valid(), *4, 5*, *8*, *14*, *44–46*

require_any, 9
require_any(), *3*, *13*, *18, 19*, *42*
require_between, 10
require_between(), *14*, *17*, *26*, *31*, *36*, *38*, *40, 41*
require_character, 11
require_character(), *12*, *15*, *23*, *27*, *35*
require_class, 11
require_class(), *11*, *15*, *23*, *27*, *35*
require_col, 12
require_col(), *3*, *9*, *18, 19*, *42*
require_contains, 13
require_contains(), *10*, *17*, *26*, *31*, *36*, *38*, *40, 41*
require_custom, 14
require_custom(), *4, 5*, *7, 8*, *44–46*
require_df, 15
require_df(), *11, 12*, *23*, *27*, *35*
require_dim, 16
require_dim(), *12*, *22*, *24–26*, *28–31*, *34, 35*, *39, 40*, *42*
require_dir_exists, 16
require_dir_exists(), *20*, *38*, *43*
require_disjoint, 17
require_disjoint(), *10*, *14*, *26*, *31*, *36*, *38*, *40, 41*
require_each, 18
require_each(), *3*, *9*, *13*, *19*, *42*
require_fields, 19
require_fields(), *3*, *9*, *13*, *18*, *42*
require_file_exists, 20
require_file_exists(), *17*, *38*, *43*
require_finite, 21
require_finite(), *33*
require_function, 21
require_function(), *14*
require_has_cols, 22
require_has_cols(), *16*, *24–26*, *28–31*, *34, 35*, *39, 40*, *42*
require_integer, 23
require_integer(), *11, 12*, *15*, *27*, *35*
require_length, 23
require_length(), *16*, *22*, *24–26*, *28–31*, *34, 35*, *39, 40*, *42*
require_length_matches, 24
require_length_matches(), *16*, *22*, *24–26*, *28–31*, *34, 35*, *39, 40*, *42*
require_length_max, 25
require_length_max(), *16*, *22*, *24*, *26*, *28–31*, *34, 35*, *39, 40*, *42*
require_length_min, 25
require_length_min(), *16*, *22*, *24, 25*, *28–31*, *34, 35*, *39, 40*, *42*
require_levels, 26
require_levels(), *10*, *14*, *17*, *31*, *36*, *38*, *40, 41*
require_logical, 27
require_logical(), *11, 12*, *15*, *23*, *35*
require_named, 27
require_named(), *16*, *22*, *24–26*, *29–31*, *34, 35*, *39–42*
require_names, 28
require_names(), *16*, *22*, *24–26*, *28*, *30, 31*, *34, 35*, *39, 40*, *42*
require_nchar, 29
require_nchar(), *32*, *37*
require_ncol_matches, 30
require_ncol_matches(), *16*, *22*, *24–26*, *28, 29*, *31*, *34, 35*, *39, 40*, *42*
require_ncol_min, 30
require_ncol_min(), *16*, *22*, *24–26*, *28–30*, *34, 35*, *39, 40*, *42*
require_negative, 31
require_negative(), *10*, *14*, *17*, *26*, *36*, *38*, *40, 41*
require_no_na, 33
require_no_na(), *10*, *17*, *20, 21*, *29*, *31–33*, *36–38*, *40*
require_nonempty, 32
require_nonempty(), *29*, *37*
require_not_null, 32
require_not_null(), *21*, *33*
require_nrow_matches, 34
require_nrow_matches(), *16*, *22*, *24–26*, *28–31*, *35*, *39, 40*, *42*
require_nrow_min, 34
require_nrow_min(), *16*, *22*, *24–26*, *28–31*,

34, *39, 40*, *42*
require_numeric, 35
require_numeric(), *11, 12*, *15*, *23*, *27*
require_one_of, 36
require_one_of(), *10*, *13, 14*, *17*, *26*, *31*, *38*, *40, 41*
require_pattern, 36
require_pattern(), *29*, *32*
require_positive, 37
require_positive(), *10*, *14*, *17*, *26*, *31*, *36*, *40, 41*
require_readable, 38
require_readable(), *17*, *20*, *43*
require_scalar, 39
require_scalar(), *16*, *22*, *24–26*, *28–31*, *34, 35*, *40*, *42*
require_set_equal, 39
require_set_equal(), *10*, *14*, *17*, *26*, *31*, *36*, *38*, *41*
require_sorted, 40
require_sorted(), *16*, *22*, *24–26*, *28–31*, *34, 35*, *39*, *42*
require_unique, 41
require_unique(), *10*, *14*, *17*, *26*, *31*, *36*, *38*, *40*
require_unique_names, 41
require_unique_names(), *16*, *22*, *24–26*, *28–31*, *34, 35*, *39, 40*
require_valid, 42
require_valid(), *3*, *9*, *13*, *18, 19*
require_writable, 43
require_writable(), *17*, *20*, *38*
restrict, 44
restrict(), *4–6*, *8*, *14*, *45, 46*

steps, 45
steps(), *4, 5*, *8*, *14*, *44*, *46*

tryCatch(), *8*

validation_errors, 46
validation_errors(), *4, 5*, *7, 8*, *14*, *44, 45*