Package: fluffy (via r-universe)

June 11, 2026

Title Schema-Based Validation of 'R' Objects with User-Defined Rules

Version 1.0.0.9000

Description A schema-based validation framework for 'R' objects using user-defined rules. Provides three 'S7' classes 'Registry', 'Schema', and 'Validator' to manage rules, define list-based schemas, and validate data in a flexible and extensible manner.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Suggests jsonlite, kableExtra, knitr, lobstr, readr, rmarkdown, stats, testthat (>= 3.0.0), yaml

Config/testthat/edition 3

Imports methods, S7, utils

URL https://lj-jenkins.github.io/fluffy/, https://github.com/LJ-Jenkins/fluffy

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

BugReports https://github.com/LJ-Jenkins/fluffy/issues

Repository https://lj-jenkins.r-universe.dev

Date/Publication 2026-06-10 09:14:42 UTC

RemoteUrl https://github.com/lj-jenkins/fluffy

RemoteRef HEAD

RemoteSha d5b4018474b80cbc59f62e0ad062669d4c424b8c

Contents

`add_rule` *Add Rules to a Registry*

Description

Add validator, schema, (schema) cross, type, or coerce rules to a Registry, Schema, or Validator object.

Usage

```
add_rule(
  obj,
  name,
  validator_fn,
  schema_fn = NULL,
  rule_type = c("validate", "control", "transform", "finalize")
)

add_cross_rule(obj, name, rule_names, cross_fn)

add_type_rule(obj, type_name, type_fn)

add_coerce_rule(obj, coerce_name, coerce_fn)
```

Arguments

`obj`	`Registry`, `Schema`, or `Validator` object.
`name`	String specifying the name of the rule to add. Rule names cannot be the same as any existing rule names.
`validator_fn`	Function that validates a data field against a schema field. Every validator rule function must return `NULL` or a named list. A `NULL` return indicates validation success with no transformed data or alteration to the control flow. A named list return can accept the following elements: • `data`: the (optionally transformed) data value to be reassigned and passed to subsequent rules. If not returned, the original data value is used. • `error`: a string error message if validation fails. If not returned, validation is assumed to have succeeded. • `continue`: boolean indicating whether to continue validating subsequent rules in the schema node. If not returned, assumed to be `TRUE`.
`schema_fn`	Function to validate a schema field definition. If `NULL`, a default function that performs no validation is used. `Schema` validation functions should return `NULL` if the schema field is valid, and a string error message if invalid.
`rule_type`	String specifying the type of rule to add. Must be one of `"control"`, `"transform"`, `"validate"`, `"finalize"`.

`rule_names` Character vector specifying the names of the existing schema rules that the cross rule will operate over. Must be of length 2 or more, and all names must be of existing schema rules in the `Registry`.

`cross_fn` Function to validate schema field definitions across multiple schema rules. `Schema` cross functions should return `NULL` if the schema fields are valid, and a string error message if invalid.

`type_name, coerce_name` String specifying the name of the type/coerce rule value to add.

`type_fn, coerce_fn` Function to validate/coerce a field value.

Details

To be able to correctly pass the required arguments for the data/schema validations, rule functions must have the following argument semantics:

type/coerce rule functions:

- `function(field)`

schema/cross rule functions:

- `function(field, ...)`
- `function(field, .self, ...)|function(field, .schema, ...)`
- `function(field, .self, .schema)`

validator rule functions:

- `function(field, schema_field, ...)`
- `function(field, schema_field, .self, ...)|function(field, schema_field, .data, ...)`
- `function(field, schema_field, .self, .data)`

Note: `field` and `schema_field` are positional, and thus can be named anything, whilst `.self` (the object itself: `Schema` for schema rules, `Validator` for validator rules) and `.schema`/`.data` (the full schema/data in the respective walks) are named arguments and must be named as such.

Despite Registry using environments to store rules, which are mutable, add_rule methods copy the existing environment and the new rule into a new environment, meaning that the original object is not modified. See examples.

For full details see the vignettes on adding custom rules.

For more information on validating data with a `Schema`, see the vignettes on builtin rules and data validation.

Value

The input object with the new rule added to the associated `Registry`.

See Also

Registry, Schema, and Validator classes.

Examples

```
s <- Schema(list(check_my_attr = 1L))
s@errors # doesn't recognise rule

s <- add_rule(
  obj = s,
  name = "check_my_attr",
  validator_fn = function(data_field, schema_field, ...) {
    if (attr(data_field, "my_attr") != schema_field) {
      list(error = "Data doesn't match schema 'my_attr'.")
    }
  },
  schema_fn = function(schema_field, ...) {
    if (!is.character(schema_field) || length(schema_field) != 1L) {
      "Must be length 1 character"
    }
  },
  rule_type = "validate"
)

# rule recognised and schema automatically re-validated
s@errors

# validation works with the new rule
s@schema$check_my_attr <- "Hi"
Validator(structure(1L, my_attr = "Hi"), s)@valid # TRUE

# schema cross rules invalidate when constituent rules are invalid
s <- add_cross_rule(
  obj = s,
  name = "min_and_max_val_add_to_10",
  rule_names = c("min_val", "max_val"),
  cross_fn = function(schema_field, ...) {
    if (schema_field$min_val + schema_field$max_val == 10) {
      "`min_val` and `max_val` cannot add to 10."
    }
  }
)
s@schema <- list(min_val = 2, max_val = 8)
s@errors # cross rule error

r <- Registry()
r <- add_type_rule(r, "my_type", function(x) {
  inherits(x, "my_type")
})
s <- Schema(list(type = "my_type"), registry = r)
s@valid # TRUE
Validator(structure(1, class = "my_type"), s)@valid # TRUE

r <- add_coerce_rule(r, "my_type", function(x) {
  structure(x, class = c("my_type", class(x)))
})
```

```
s <- Schema(list(coerce = "my_type"), registry = r)
v <- Validator(1, s)
class(v@data) # "my_type" "numeric"

# environments are copied, so original registry is not modified
r <- Registry()
r2 <- add_type_rule(r, "my_type", function(x) x)
identical(r@type_map, r2@type_map) # FALSE
```

`Registry` *Registry*

Description

Create a `Registry` object that defines and stores rules for validating data against a Schema.

Usage

```
Registry()

is.Registry(x)
```

Arguments

`x`	Object to be tested.

Details

The `Registry` class serves as a central repository for all the rules used in `fluffy` schema and data validation. It is passed to the Schema and Validator classes, which use the rules stored in the `Registry` to validate schemas and data, respectively.

To see the available builtin rules, use the helper function `show_builtins()`, which lists all the builtin rules in a `Registry`.

As `Registry` objects are automatically created in the `Schema` constructor, which is in turn called in the `Validator` constructor, a `Registry` does not need to be created separately to use `fluffy`'s validation functionality. Custom rules can also be added to the `Registry` within a `Schema` or `Validator` object. However, rule addition will trigger re-validation of the given object, so for the addition of many new rules, it is beneficial to create a `Registry` object, add all the rules to it, and then pass it to the other classes.

For full details see the vignettes on builtin rules, data validation, and adding custom rules.

Value

A `Registry` S7 object.

Additional properties

`@rule_names` (Read-only) All rules, in order of evaluation. Derived from the combination of the `control`, `transform`, `validate`, and `finalize` rules.

`@control_rules` Rules that influence the control flow. These will be applied in the first pass when validating data.

`@transform_rules` Rules that transform data. These will be applied in the second pass when validating data.

`@validate_rules` Rules that validate data. These will be applied in the penultimate pass when validating.

`@finalize_rules` Rules that finalize validation. These will be applied in the last pass when validating, after all other rules have been applied. These rules will only apply if no previous rules for that schema node have failed.

`@str_to_fn_rules` Schema rules that are allowed to have string or function values, with string values being converted to functions automatically during schema validation.

`@str_to_fn_converter` Function that converts string values to functions for the `@str_to_fn_rules`.

`@type_names` (Read-only) Allowed type names. Derived from the keys of the `@type_map`.

`@type_map` Environment mapping type names to type definition functions. Can only be added to via `add_type_rule()`.

`@coerce_names` (Read-only) Allowed coercion names. Derived from the keys of the `@coerce_map`.

`@coerce_map` Environment mapping coercion names to coercion functions. Can only be added to via `add_coerce_rule()`.

`@schema_rules` Environment mapping schema rule names to schema validation functions. Can only be added to via `add_rule()`.

`@cross_rule_names` (Read-only) Schema cross rules. Derived from the keys of the `@cross_rules` environment.

`@cross_rules` Environment containing the schema cross rule functions (rules that check relationships between multiple schema rule values). Can only be added to via `add_cross_rule()`.

`@validator_rules` Environment mapping validator rule names to validator functions. Can only be added to via `add_rule()`.

See Also

Schema and Validator constructors. add_rule for adding rules to a `Registry`.

Examples

```
r <- Registry()
s <- Schema(list(type = "integer"), registry = r)

is.Registry(r)
```

Schema *Schema*

Description

Create a `Schema` object that defines and validates a schema for the future validation of R objects using a `Validator`.

Usage

```
Schema(schema, registry = Registry(), error = FALSE, ...)

is.Schema(x)
```

Arguments

`schema`	Non-empty list defining the schema. Each leaf element to be validated must be named with rules that are registered in the schema rule registry (see Registry and add_rule). Named elements must be unique at a given level of the list.
`registry`	Object of class Registry containing the schema and validator rules to use for schema validation.
`error`	Single logical value. If `TRUE`, the constructor throws an error when the schema is invalid.
`...`	Named arguments passed to internal error message formatting: `max_depth`, `max_width`, `max_rows`, and `UTF8` which control the truncation and printing of the error message when the schema validation fails. `max_*` arguments accept integerish scalar values, and `UTF8` accepts a single logical value. Unnamed arguments or named arguments other than the above will be ignored.
`x`	Object to be tested.

Details

The `Schema` class defines the structure and rules for validating data using a `Validator`. It ingests the input schema list and then reorders and validates the schema according to the Registry. The resulting `@schema` property defines the validation behaviour and data matching.

Each element of the input list represents a field with named rules (e.g., `'type'`, `'required'`, `'min_val'`) that are checked against the schema rule registry. Cross rules (e.g., `'min_val_larger_than_max_val'`, where `'min_val' < 'max_val'`) are also evaluated when all constituent rules pass individually. See the Registry class for details.

The schema is re-evaluated upon any change to the schema properties, with the `@valid` property indicating whether the schema is valid (all rules pass) or invalid (any rule fails). If invalid, when passed to a Validator the validation will fail immediately.

For full details see the vignettes on builtin rules, data validation, and adding custom rules.

Value

A Schema S7 object.

Additional properties

`@schema` The input schema list, with nodes reordered to match the order of the rules in the registry, and any strings converted to functions where applicable.

`@errors` (Read-only) A list mirroring the schema structure, with `NULL` for valid rules and character error messages for invalid rules.

`@Registry` An object of class Registry containing the schema and validator rules used for validation.

`@.schema_cache` (Read-only) An environment for caching schema validation results. Used to avoid redundant validation.

`@error` Boolean; whether to error on invalid schemas.

`@error_print_opts` A list of options for error message printing when `error = TRUE`. These options are also used by the `Validator` class that ingests the `Schema`. Defaults are

- `max_depth = 10L`
- `max_width = getOption("width")`
- `max_rows = 30L`
- `UTF8 = l10n_info()[["UTF-8"]]`

`@valid` (Read-only) Boolean; `TRUE` if schema is valid, `FALSE` otherwise.

See Also

add_rule for adding rules to a `Registry`. Validator for validating data against a `Schema`.

Examples

```
# A valid schema
s <- Schema(list(
  name = list(type = "character", required = TRUE),
  age = list(type = "numeric", min_val = 0, max_val = 150)
))
s@valid # TRUE

# An invalid schema (type must be a string)
s <- Schema(list(name = list(type = 123)))
s@valid # FALSE
s@errors # error message

# To error on invalid schema
try(Schema(list(name = list(type = 123)), error = TRUE))
```

show_builtins *Show Builtin Rules*

Description

Helper to show the builtin rules in a `Registry`, and their behaviour.

Usage

```
show_builtins(rules = c("all", "validation", "cross"))
```

Arguments

`rules`	Which builtin rules to show. `"validation"` for the standard schema/data validation rules, `"cross"` for the cross rules that check for consistency between rules, or `"all"` to show both.

Value

List of data.fames if `rules = "all"`, otherwise a single data.frame, with information on the builtin rules for the rule type specified. The data.frame(s) have an attached class `fluffy_rule_info`.

Note

The `fluffy_rule_info` class has a custom print method that formats the data.frame(s) in a more readable way. The output of this function is only intended to be used for display/interactive purposes.

See Also

Registry and add_rule.

Examples

```
show_builtins()
```

Validator *Validator*

Description

Create a `Validator` object that validates/transforms R objects using a Schema.

Usage

```
Validator(data, schema, error = FALSE)

is.Validator(x)
```

Arguments

data	An R object. If using a nested schema, the object must have a [[method.
schema	Schema object or a non-empty list defining the schema. See the Schema constructor for details on the schema structure and rules.
error	Single logical value. If TRUE, the constructor throws an error when validation fails.
x	Object to be tested.

Details

The Validator class validates/transforms data using a Schema. The process checks each field in the schema against the data, applying the rules specified. Any data element not present in the schema is ignored. Validation is done in four passes, according to rule categories in the Registry: control, transform, validate, and finalize.

Validation results are stored in the @errors property, and the overall validity is indicated by the @valid property. If the @error property/argument is set to TRUE, any validation failure will result in an error being thrown. To customise the truncation of error messages, see the Schema @error_print_opts property. The (possibly) transformed data is stored in the @data property.

The Validator is re-evaluated upon any change to the object properties. If an invalid schema is provided, the validation will fail immediately with the validation error indicating that the schema is invalid. To see the errors in the schema validation, see the Validator@Schema@errors property.

For full details see the vignettes on builtin rules, data validation, and adding custom rules.

Value

A Validator S7 object.

Additional properties

@data The validated input data, with any changes that were made during validation.

@Schema An object of class Schema containing the schema used for validation, as well as the Registry.

@errors (Read-only) A list mirroring the schema structure, with NULL for rules that pass and character error messages for rules that fail.

@.validator_cache (Read-only) An environment for caching validation results. Used to avoid redundant validation.

@error Boolean; whether to error upon failure (invalid data or schema).

@valid (Read-only) Boolean; TRUE if the schema is valid and all data validation rules pass, FALSEotherwise.

See Also

Schema and Registry classes, and add_rule for adding custom rules.

Examples

```
v <- Validator(
  data = list(name = "Alice", age = 30),
  schema = list(
    name = list(type = "character", required = TRUE),
    age = list(type = "numeric", min_val = 0, max_val = 150)
  )
)
v@valid # TRUE

# Schema object can be given directly
s <- Schema(list(a = list(type = "numeric"), b = list(type = "character")))
v <- Validator(list(a = "Hello", b = 42), s)
v@valid # FALSE
v@errors

# To error on invalid schema or data
try(Validator(list(a = "Hello", b = 42), s, error = TRUE))

# Invalid schemas show their errors
try(Validator(list(42), list(type = 123), error = TRUE))

# Transforms data according to rules
v <- Validator(1, list(coerce = "character"))
v@data # "1"
```

Index