# Package: vrule (via r-universe)

July 10, 2026

**Version** 0.1.20260710

**Date** 2026-07-10

**Title** Tools to validate data based on business validation rules

**Maintainer** Emmanuel Blondel <emmanuel.blondel1@gmail.com>

**Depends** R (>= 4.0), methods

**Imports** R6, jsonlite, yaml, readr, stringr, rhandsontable, data.table, parallel

**Suggests** testthat, waldo

**Description** Provides tools to validate data based on business validation rules

**License** MIT + file LICENSE

**URL** https://github.com/fdiwg/vrule, https://fdiwg.github.io/vrule/

**BugReports** https://github.com/fdiwg/vrule/issues

**RoxygenNote** 7.3.3

**Config/pak/sysreqs** cmake make libicu-dev libuv1-dev libx11-dev

**Repository** https://fdiwg.r-universe.dev

**Date/Publication** 2026-07-10 11:49:16 UTC

**RemoteUrl** https://github.com/fdiwg/vrule

**RemoteRef** HEAD

**RemoteSha** 604856964587add0b7530df05bba8f245148ac3b

## Contents

---

`column_spec` *column_spec*

---

## Description

column_spec

column_spec

## Public fields

`name` name

`urn` urn

`dimension` dimension

`aliases` aliases

`required` required

`rules` rules

## Methods

### Public methods:

- `column_spec$new()`
- `column_spec$setName()`
- `column_spec$setURN()`
- `column_spec$isDimension()`
- `column_spec$setAliases()`
- `column_spec$setRequired()`
- `column_spec$addRule()`
- `column_spec$validate()`
- `column_spec$hasCodelist()`
- `column_spec$clone()`

**Method** `new()`**:** Initializes a column specification

*Usage:*

```
column_spec$new(obj = NULL)
```

*Arguments:*

`obj` object of class list

**Method** `setName()`**:** set name

*Usage:*

```
column_spec$setName(name)
```

*Arguments:*

`name` name

**Method** `setURN()`**:** set URN

*Usage:*

```
column_spec$setURN(urn)
```

*Arguments:*

`urn` urn

**Method** `isDimension()`**:** Set if the column is a dimension

*Usage:*

```
column_spec$isDimension(isDimension)
```

*Arguments:*

`isDimension` isDimension

**Method** `setAliases()`**:** Set aliases

*Usage:*

```
column_spec$setAliases(aliases)
```

*Arguments:*

`aliases` aliases

**Method** `setRequired()`**:** Set if the column is required

*Usage:*

```
column_spec$setRequired(required)
```

*Arguments:*

`required` required

**Method** `addRule()`**:** Adds a validation rule

*Usage:*

```
column_spec$addRule(rule)
```

*Arguments:*

`rule` rule

**Method** `validate()`**:** Method to validate column data

*Usage:*

```
column_spec$validate(values, rows)
```

*Arguments:*

`values` values

`rows` rows

*Returns:* a validation report, object of class vrule_report

**Method** `hasCodelist()`**:** Indicates if column specification includes a codelist validation rule

*Usage:*

```
column_spec$hasCodelist()
```

*Returns:* `TRUE` if it includes codelist validation, `FALSE` otherwise

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
column_spec$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## create_vrule_report *create_vrule_report*

### Description

create_vrule_report

### Usage

```
create_vrule_report(valid, category, rule, type, message)
```

### Arguments

| | |
|---|---|
| valid | valid TRUE or FALSE |
| category | category |
| rule | rule |
| type | type |
| message | message |

## decodeVrules *decodeVrules*

### Description

decodeVrules

### Usage

```
decodeVrules(json)
```

### Arguments

| | |
|---|---|
| json | json object |

format_spec *format_spec*

## Description

format_spec

format_spec

## Public fields

`name` format name

`urn` urn

`title` title

`type` tpe

`column_specs` column specifications

## Methods

### Public methods:

- `format_spec$new()`
- `format_spec$setName()`
- `format_spec$setURN()`
- `format_spec$setTitle()`
- `format_spec$setType()`
- `format_spec$addColumnSpec()`
- `format_spec$getColumnSpecByName()`
- `format_spec$getColumnSpecByURN()`
- `format_spec$getColumnSpecByAlias()`
- `format_spec$getColumnSpec()`
- `format_spec$validateStructure()`
- `format_spec$validateSeries()`
- `format_spec$validateContent()`
- `format_spec$validate()`
- `format_spec$display_as_handsontable()`
- `format_spec$validate_and_display_as_handsontable()`
- `format_spec$standardizeStructure()`
- `format_spec$standardizeContent()`
- `format_spec$createTemplate()`
- `format_spec$clone()`

**Method** `new()`**:** Initializes a format specification from a JSON list object

*Usage:*

```
format_spec$new(file = NULL, obj = NULL, json = NULL)
```

*Arguments:*

`file` a file, either in `JSON` or `YAML` format

`obj` an object of class list

`json` an object of class list. Deprecated. Use `obj` instead

**Method** `setName()`**:** Set name

*Usage:*

```
format_spec$setName(name)
```

*Arguments:*

`name` name

**Method** `setURN()`**:** Set URN

*Usage:*

```
format_spec$setURN(urn)
```

*Arguments:*

`urn` urn

**Method** `setTitle()`**:** Set title

*Usage:*

```
format_spec$setTitle(title)
```

*Arguments:*

`title` title

**Method** `setType()`**:** Set type

*Usage:*

```
format_spec$setType(type)
```

*Arguments:*

`type` type

**Method** `addColumnSpec()`**:** Adds a column specification

*Usage:*

```
format_spec$addColumnSpec(column_spec)
```

*Arguments:*

`column_spec` an object of class column_spec

**Method** `getColumnSpecByName()`**:** Get column specification by name

*Usage:*

```
format_spec$getColumnSpecByName(name)
```

*Arguments:*

`name` name

*Returns:* an object of class column_spec, or `NULL` if no column specification is found

**Method** `getColumnSpecByURN()`: Get column specification by URN

*Usage:*

```
format_spec$getColumnSpecByURN(urn)
```

*Arguments:*

`urn` urn

*Returns:* an object of class column_spec, or `NULL` if no column specification is found

**Method** `getColumnSpecByAlias()`: Get column specification by alias

*Usage:*

```
format_spec$getColumnSpecByAlias(alias)
```

*Arguments:*

`alias` alias

*Returns:* an object of class column_spec, or `NULL` if no column specification is found

**Method** `getColumnSpec()`: Get column specification

*Usage:*

```
format_spec$getColumnSpec(column)
```

*Arguments:*

`column` column name or alias

*Returns:* an object of class column_spec, or `NULL` if no column specification is found

**Method** `validateStructure()`: Applies data structure validation

*Usage:*

```
format_spec$validateStructure(data)
```

*Arguments:*

`data` object of class data.frame or tibble

*Returns:* an object of class data.frame if any data structure validation issue is found (ERROR or WARNING), or `NULL` if valid

**Method** `validateSeries()`: Applies data series validation

*Usage:*

```
format_spec$validateSeries(data)
```

*Arguments:*

`data` object of class data.frame or tibble

*Returns:* an object of class data.frame if any data series validation issue is found (ERROR or WARNING), or `NULL` if valid

**Method** `validateContent()`: Applies data content validation

*Usage:*

```
format_spec$validateContent(
  data,
  mode = c("column", "pair"),
  parallel = FALSE,
  ...
)
```

*Arguments:*

`data` object of class data.frame or tibble

`mode` validation mode, either "column" (default) or "pair"

`parallel` whether the validation should be run as parallel (default `FALSE`)

`...` any other arg

*Returns:* an object of class data.frame

**Method** `validate()`**:** Applies data validation

*Usage:*

```
format_spec$validate(data, mode = c("column", "pair"), parallel = FALSE, ...)
```

*Arguments:*

`data` object of class data.frame or tibble

`mode` validation mode, either "column" (default) or "pair"

`parallel` whether the validation should be run as parallel (default `FALSE`)

`...` any other arg

*Returns:* an object of class data.frame

**Method** `display_as_handsontable()`**:** Display data and validation report as Handsontable

*Usage:*

```
format_spec$display_as_handsontable(data, report, ...)
```

*Arguments:*

`data` data

`report` report

`...` any other arg

*Returns:* an object of class rhandsontable

**Method** `validate_and_display_as_handsontable()`**:** Applies data validation

*Usage:*

```
format_spec$validate_and_display_as_handsontable(
  data,
  parallel = FALSE,
  read_only = TRUE,
  use_css_classes = FALSE,
  ...
)
```

*Arguments:*

`data` object of class data.frame or tibble

`parallel` whether the validation should be run as parallel (default `FALSE`)

`read_only` read only

`use_css_classes` use css classes

`...` any other arg

*Returns:* an object of class rhandsontable

**Method** `standardizeStructure()`**:** Standardize structure

*Usage:*

```
format_spec$standardizeStructure(data, exclude_unused = TRUE)
```

*Arguments:*

`data` data

`exclude_unused` exclude unsed columns

*Returns:* the standardized data structure

**Method** `standardizeContent()`**:** Standardize content

*Usage:*

```
format_spec$standardizeContent(data)
```

*Arguments:*

`data` data

*Returns:* the standardized data

**Method** `createTemplate()`**:** Creates template based on the format specification, including a template structure and eventual reference data files (codelists)

*Usage:*

```
format_spec$createTemplate(use_aliases = FALSE, dir)
```

*Arguments:*

`use_aliases` use aliases?

`dir` directory where to save the template

*Returns:* the path of the output template (as ZIP file)

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
format_spec$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

getClassesInheriting *getClassesInheriting*

## Description

get the list of classes inheriting a given super class provided by its name

## Usage

```
getClassesInheriting(classname, extended, pretty)
```

## Arguments

| | |
|---|---|
| classname | the name of the superclass for which inheriting sub-classes have to be listed |
| extended | whether we want to look at user namespace for third-party sub-classes |
| pretty | prettify the output as data.frame |

## Examples

```
getClassesInheriting("vrule_integer")
```

getVruleClasses *getVruleClasses*

## Description

get the list of validation classes, ie classes extending vrule_abstract super class, including classes eventually defined outside **vrule**. In case the latter is on the search path, the list of validation classes will be cached for optimized used by **vrule** encoder/decoder.

## Usage

```
getVruleClasses()
```

## Author(s)

Emmanuel Blondel, <emmanuel.blondel1@gmail.com>

## Examples

```
getVruleClasses()
```

`getVruleOption` *getVruleOption*

## Description

getVruleOption

## Usage

```
getVruleOption(opt)
```

## Arguments

| | |
|---|---|
| `opt` | an option name |

## Value

the value for that option

`getVruleOptions` *getVruleOptions*

## Description

getVruleOptions

## Usage

```
getVruleOptions()
```

`setVruleOptions` *setVruleOptions*

## Description

setVruleOptions

## Usage

```
setVruleOptions(...)
```

## Arguments

| | |
|---|---|
| `...` | options |

## vrule *Tools to validate data based on business validation rules*

### Description

Provides tools to validate data based on business validation rules.

### Author(s)

**Maintainer**: Emmanuel Blondel <emmanuel.blondel@fao.org> (ORCID)

Authors:

- Alexandre Bennici <alexandre.bennici@fao.org> (ORCID)

### See Also

Useful links:

- https://github.com/fdiwg/vrule
- https://fdiwg.github.io/vrule/
- Report bugs at https://github.com/fdiwg/vrule/issues

## vrule_abstract *Abstract validation rule*

### Description

Abstract validation rule

Abstract validation rule

### Methods

#### Public methods:

- vrule_abstract$new()
- vrule_abstract$validate()
- vrule_abstract$getType()
- vrule_abstract$getCategory()
- vrule_abstract$getName()
- vrule_abstract$clone()

**Method** `new()`**:** Initializes an abstract validation rule

*Usage:*

```
vrule_abstract$new(..., type = c("ERROR", "WARNING"))
```

*Arguments:*

`...` args

`type` the type of rule either `ERROR` or `WARNING`

**Method** `validate()`**:** Abstract method to validate data

*Usage:*

```
vrule_abstract$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `getType()`**:** Get validation type

*Usage:*

```
vrule_abstract$getType()
```

*Returns:* `ERROR` or `WARNING`

**Method** `getCategory()`**:** Get validation rule category

*Usage:*

```
vrule_abstract$getCategory()
```

*Returns:* the validation rule category

**Method** `getName()`**:** Get validation rule name

*Usage:*

```
vrule_abstract$getName()
```

*Returns:* the validation rule name

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_abstract$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

vrule_abstract_complex *vrule_abstract_complex*

## Description

vrule_abstract_complex

vrule_abstract_complex

## Super class

`vrule::vrule_abstract` -> `vrule_abstract_complex`

## Methods

### Public methods:

- `vrule_abstract_complex$new()`
- `vrule_abstract_complex$validate()`
- `vrule_abstract_complex$clone()`

**Method** `new()`**:** Initializes an abstract complex validation rule

*Usage:*

```
vrule_abstract_complex$new(...)
```

*Arguments:*

`...` args

**Method** `validate()`**:** Abstract method to validate data

*Usage:*

```
vrule_abstract_complex$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_abstract_complex$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

vrule_abstract_simple *vrule_abstract_simple*

## Description

vrule_abstract_simple

vrule_abstract_simple

## Super class

vrule::vrule_abstract -> vrule_abstract_simple

## Methods

### Public methods:

- vrule_abstract_simple$new()
- vrule_abstract_simple$validate()
- vrule_abstract_simple$clone()

**Method** `new()`**:** Initializes an abstract simple validation rule

*Usage:*

```
vrule_abstract_simple$new(...)
```

*Arguments:*

`...` args

**Method** `validate()`**:** Abstract method to validate data

*Usage:*

```
vrule_abstract_simple$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_abstract_simple$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

vrule_character *vrule_character*

## Description

vrule_character

vrule_character

## Super classes

vrule::vrule_abstract -> vrule::vrule_abstract_simple -> vrule::vrule_datatype -> vrule_character

## Methods

### Public methods:

- vrule_character$new()
- vrule_character$validate()
- vrule_character$clone()

**Method** `new()`**:** Initializes a character data validation rule

*Usage:*

```
vrule_character$new(na_allowed = FALSE, ...)
```

*Arguments:*

`na_allowed` is NA allowed?

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_character$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_character$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

vrule_codelist *vrule_codelist*

## Description

vrule_codelist

vrule_codelist

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_simple` -> `vrule_codelist`

## Public fields

`ref_data_url` ref data url

`ref_data` ref data

`ref_data_column` ref data column

`ref_data_column_alt` ref data alternate column

`ref_meta_url` ref metadata url

`ref_meta` ref metadata

`allow_labels` allow labels?

## Methods

### Public methods:

- `vrule_codelist$new()`
- `vrule_codelist$validate()`
- `vrule_codelist$clone()`

**Method** `new()`**:** Initializes a codelist validation rule

*Usage:*

```
vrule_codelist$new(
  ref_data_url = NULL,
  ref_data_column = "code",
  ref_data_column_alt = "label",
  allow_labels = FALSE,
  ref_meta_url = NULL,
  ...
)
```

*Arguments:*

`ref_data_url` ref data url

`ref_data_column` ref data column

`ref_data_column_alt` ref data alternate column

`allow_labels` Allow labels?

`ref_meta_url` ref metadata url

`...` any other arg

**Method** `validate()`**:** Validates value with a codelist validation rule

*Usage:*

```
vrule_codelist$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_codelist$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_cross_column` *vrule_cross_column*

---

## Description

vrule_cross_column

vrule_cross_column

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_complex` -> `vrule_cross_column`

## Public fields

`operator` operator

`expr` expr

## Methods

### Public methods:

- `vrule_cross_column$new()`
- `vrule_cross_column$validate()`
- `vrule_cross_column$clone()`

**Method** `new()`**:** Initializes a cross-column check validation rule

*Usage:*

```
vrule_cross_column$new(operator, expr, ...)
```

*Arguments:*

`operator` operator

`expr` expr

`...` any other arg

**Method** `validate()`**:** Abstract method to validate data

*Usage:*

```
vrule_cross_column$validate(value, row)
```

*Arguments:*

`value` value

`row` row

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_cross_column$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_datatype` *vrule_datatype*

---

## Description

vrule_datatype

vrule_datatype

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_simple` -> `vrule_datatype`

## Public fields

`datatype` data type

`na_allowed` is NA allowed?

## Methods

### Public methods:

- vrule_datatype$new()
- vrule_datatype$validate()
- vrule_datatype$clone()

**Method** `new()`**:** Initializes a data type validation rule

*Usage:*

```
vrule_datatype$new(datatype, na_allowed = FALSE, ...)
```

*Arguments:*

`datatype` data type

`na_allowed` is NA allowed?

`...` any other arg

**Method** `validate()`**:** Abstract method to validate data

*Usage:*

```
vrule_datatype$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_datatype$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_date` *vrule_date*

---

## Description

vrule_date

vrule_date

## Super classes

vrule::vrule_abstract -> vrule::vrule_abstract_simple -> vrule::vrule_datatype -> vrule::vrule_character -> vrule_date

## Methods

### Public methods:

- vrule_date$new()
- vrule_date$validate()
- vrule_date$clone()

**Method** `new()`**:** Initializes a date validation rule

*Usage:*

```
vrule_date$new(na_allowed = FALSE, ...)
```

*Arguments:*

`na_allowed` `TRUE` if NA values are allowed, `FALSE` `otherwise`

`...` any other arg

**Method** `validate()`**:** Validates a date

*Usage:*

```
vrule_date$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_date$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_date_max` *vrule_date_max*

---

## Description

vrule_date_max

vrule_date_max

## Details

vrule_date_max

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule::vrule_operator_relational` -> `vrule_date_max`

## Methods

### Public methods:

- vrule_date_max$new()
- vrule_date_max$validate()
- vrule_date_max$clone()

**Method** `new()`**:** Initializes a date max validation rule

*Usage:*

```
vrule_date_max$new(maxValue, ...)
```

*Arguments:*

`maxValue` max value

`...` any other arg

**Method** `validate()`**:** Validates data based on a max date threshold

*Usage:*

```
vrule_date_max$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_date_max$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_date_min` *vrule_date_min*

---

## Description

vrule_date_min

vrule_date_min

## Details

vrule_date_min

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule::vrule_operator_relational` -> `vrule_date_min`

## Methods

### Public methods:

- vrule_date_min$new()
- vrule_date_min$validate()
- vrule_date_min$clone()

**Method** `new()`**:** Initializes a date min validation rule

*Usage:*

```
vrule_date_min$new(minValue, ...)
```

*Arguments:*

`minValue` min value

`...` any other arg

**Method** `validate()`**:** Validates data based on a min date threshold

*Usage:*

```
vrule_date_min$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_date_min$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_date_range` *vrule_date_range*

---

## Description

vrule_date_range

vrule_date_range

## Details

vrule_date_range

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_simple` -> `vrule_date_range`

## Public fields

`minValue` min value

`maxValue` max value

## Methods

### Public methods:

- `vrule_date_range$new()`
- `vrule_date_range$validate()`
- `vrule_date_range$clone()`

**Method** `new()`**:** Initializes a range validation rule

*Usage:*

```
vrule_date_range$new(minValue, maxValue, ...)
```

*Arguments:*

`minValue` min value

`maxValue` max value

`...` any other arg

**Method** `validate()`**:** Validates data based on a date range

*Usage:*

```
vrule_date_range$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_date_range$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

`vrule_date_threshold` *vrule_date_threshold*

## Description

vrule_date_threshold

vrule_date_threshold

## Details

vrule_date_threshold

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule::vrule_operator_relational` -> `vrule_date_threshold`

## Methods

### Public methods:

- `vrule_date_threshold$new()`
- `vrule_date_threshold$validate()`
- `vrule_date_threshold$clone()`

**Method** `new()`**:** Initializes a date threshold validation rule

*Usage:*

```
vrule_date_threshold$new(operator, threshold, ...)
```

*Arguments:*

`operator` operator

`threshold` threshold

`...` any other arg

**Method** `validate()`**:** Validates data based on a date threshold

*Usage:*

```
vrule_date_threshold$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_date_threshold$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

vrule_double *vrule_double*

## Description

vrule_double

vrule_double

## Super classes

vrule::vrule_abstract -> vrule::vrule_abstract_simple -> vrule::vrule_datatype -> vrule_double

## Methods

### Public methods:

- vrule_double$new()
- vrule_double$validate()
- vrule_double$clone()

**Method** `new()`**:** Initializes a double data validation rule

*Usage:*

```
vrule_double$new(na_allowed = FALSE, ...)
```

*Arguments:*

`na_allowed` is NA allowed?

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_double$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_double$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

vrule_if *vrule_if*

## Description

vrule_if

vrule_if

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_complex` -> `vrule_if`

## Public fields

`if_condition` if expression

`then_apply` list of expressions to apply if condition is fulfilled

`else_apply` list of expressions to apply if condition is not fulfilled

## Methods

### Public methods:

- `vrule_if$new()`
- `vrule_if$validate()`
- `vrule_if$clone()`

**Method** `new()`**:** Initializes a conditionnal validation rule

*Usage:*

```
vrule_if$new(if_condition, then_apply = list(), else_apply = list(), ...)
```

*Arguments:*

`if_condition` if condition

`then_apply` list of expressions to apply if condition is fulfilled

`else_apply` list of expressions to apply if condition is not fulfilled

`...` any other arg

**Method** `validate()`**:** Abstract method to validate data

*Usage:*

```
vrule_if$validate(value, row)
```

*Arguments:*

`value` value

`row` row

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_if$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_integer` *vrule_integer*

---

## Description

vrule_integer

vrule_integer

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_simple` -> `vrule::vrule_datatype` -> `vrule_integer`

## Methods

### Public methods:

- `vrule_integer$new()`
- `vrule_integer$validate()`
- `vrule_integer$clone()`

**Method** `new()`**:** Initializes a integer data validation rule

*Usage:*

```
vrule_integer$new(na_allowed = FALSE, ...)
```

*Arguments:*

`na_allowed` is NA allowed?

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_integer$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_integer$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

vrule_logical *vrule_logical*

## Description

vrule_logical

vrule_logical

## Super classes

vrule::vrule_abstract -> vrule::vrule_abstract_simple -> vrule::vrule_datatype -> vrule_logical

## Methods

### Public methods:

- vrule_logical$new()
- vrule_logical$validate()
- vrule_logical$clone()

**Method** `new()`**:** Initializes a logical data validation rule

*Usage:*

```
vrule_logical$new(na_allowed = FALSE, ...)
```

*Arguments:*

`na_allowed` is NA allowed?

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_logical$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_logical$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

vrule_mapping *vrule_mapping*

## Description

vrule_mapping

vrule_mapping

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_complex` -> `vrule_mapping`

## Public fields

`ref_data_url` ref data url

`ref_data` ref data

`ref_source_term` ref source term

`ref_target_term` ref target term

`data_target_term` data target term

`ref_meta_url` ref meta url

`ref_meta` ref meta

## Methods

### Public methods:

- `vrule_mapping$new()`
- `vrule_mapping$validate()`
- `vrule_mapping$clone()`

**Method** `new()`**:** Initializes a mapping-based validation rule

*Usage:*

```
vrule_mapping$new(
  ref_data_url = NULL,
  ref_source_term = NULL,
  ref_target_term = NULL,
  data_target_term = NULL,
  ref_meta_url = NULL,
  ...
)
```

*Arguments:*

`ref_data_url` ref data url

`ref_source_term` ref source term

`ref_target_term` ref target term

`data_target_term` data target term

`ref_meta_url` ref metadata url

`...` any other arg

**Method** `validate()`**:** Abstract method to validate data

*Usage:*

```
vrule_mapping$validate(value, row)
```

*Arguments:*

`value` value

`row` row

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_mapping$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_max` *vrule_max*

---

## Description

vrule_max

vrule_max

## Details

vrule_max

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule::vrule_operator_relational` -> `vrule_max`

## Methods

### Public methods:

- `vrule_max$new()`
- `vrule_max$validate()`
- `vrule_max$clone()`

**Method** `new()`**:** Initializes a max threshold validation rule

*Usage:*

```
vrule_max$new(maxValue, ...)
```

*Arguments:*

`maxValue` max value

`...` any other arg

**Method** `validate()`**:** Validates data based on a max threshold

*Usage:*

```
vrule_max$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_max$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_min` *vrule_min*

---

## Description

vrule_min

vrule_min

## Details

vrule_min

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule::vrule_operator_relational` -> `vrule_min`

## Methods

### Public methods:

- `vrule_min$new()`
- `vrule_min$validate()`
- `vrule_min$clone()`

**Method** `new()`**:** Initializes a min threshold validation rule

*Usage:*

```
vrule_min$new(minValue, ...)
```

*Arguments:*

`minValue` min value

`...` any other arg

**Method** `validate()`**:** Validates data based on a min threshold

*Usage:*

```
vrule_min$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_min$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_numeric` *vrule_numeric*

---

## Description

vrule_numeric

vrule_numeric

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_simple` -> `vrule::vrule_datatype` -> `vrule_numeric`

## Methods

### Public methods:

- `vrule_numeric$new()`
- `vrule_numeric$validate()`
- `vrule_numeric$clone()`

**Method** `new()`**:** Initializes a numeric data validation rule

*Usage:*

```
vrule_numeric$new(na_allowed = FALSE, ...)
```

*Arguments:*

`na_allowed` is NA allowed?

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_numeric$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_numeric$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_operator_and` *vrule_operator_and*

---

## Description

vrule_operator_and

vrule_operator_and

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule::vrule_operator_logical` -> `vrule_operator_and`

## Methods

### Public methods:

- `vrule_operator_and$new()`
- `vrule_operator_and$validate()`
- `vrule_operator_and$clone()`

**Method** `new()`**:** Initializes a logical AND operator validation rule

*Usage:*

```
vrule_operator_and$new(...)
```

*Arguments:*

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_operator_and$validate(value, row)
```

*Arguments:*

`value` value

`row` row

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_operator_and$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_operator_binary` *vrule_operator_binary*

---

## Description

vrule_operator_binary

vrule_operator_binary

## Super class

`vrule::vrule_abstract` -> `vrule_operator_binary`

## Public fields

`operator` operator

`expr` expr

## Methods

### Public methods:

- `vrule_operator_binary$new()`
- `vrule_operator_binary$validate()`
- `vrule_operator_binary$clone()`

**Method** `new()`**:** Initializes a binary operator validation rule

*Usage:*

```
vrule_operator_binary$new(operator, expr, ...)
```

*Arguments:*

`operator` operator

`expr` expr

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_operator_binary$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other arg

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_operator_binary$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_operator_logical` *vrule_operator_logical*

---

## Description

vrule_operator_logical

vrule_operator_logical

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule_operator_logical`

## Public fields

`operator_fun` operator function

`rules` rules

## Methods

### Public methods:

- `vrule_operator_logical$new()`
- `vrule_operator_logical$validate()`
- `vrule_operator_logical$clone()`

**Method** `new()`**:** Initializes a logical operator validation rule

*Usage:*

```
vrule_operator_logical$new(operator, ...)
```

*Arguments:*

`operator` operator

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_operator_logical$validate(value, row)
```

*Arguments:*

`value` value

`row` row

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_operator_logical$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_operator_or` *vrule_operator_or*

---

## Description

vrule_operator_or

vrule_operator_or

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule::vrule_operator_logical` -> `vrule_operator_or`

## Methods

### Public methods:

- `vrule_operator_or$new()`
- `vrule_operator_or$validate()`
- `vrule_operator_or$clone()`

**Method** `new()`**:** Initializes a logical OR operator validation rule

*Usage:*

```
vrule_operator_or$new(...)
```

*Arguments:*

`...` any other arg

**Method** `validate()`**:** Method to validate data

*Usage:*

```
vrule_operator_or$validate(value, row)
```

*Arguments:*

`value` value

`row` row

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_operator_or$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_operator_relational` *vrule_operator_relational*

---

## Description

vrule_operator_relational

vrule_operator_relational

## Details

vrule_operator_relational

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule_operator_relational`

## Methods

### Public methods:

- `vrule_operator_relational$new()`
- `vrule_operator_relational$validate()`
- `vrule_operator_relational$clone()`

**Method** `new()`**:** Initializes a operator relational validation rule

*Usage:*

```
vrule_operator_relational$new(operator, expr, ...)
```

*Arguments:*

`operator` operator

`expr` expr

`...` any other arg

**Method** `validate()`**:** Abstract method to validate data

*Usage:*

```
vrule_operator_relational$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_operator_relational$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_range` *vrule_range*

---

## Description

vrule_range

vrule_range

## Details

vrule_range

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_simple` -> `vrule_range`

## Public fields

`minValue` min value

`maxValue` max value

## Methods

### Public methods:

- vrule_range$new()
- vrule_range$validate()
- vrule_range$clone()

**Method** `new()`**:** Initializes a range validation rule

*Usage:*

```
vrule_range$new(minValue, maxValue, ...)
```

*Arguments:*

`minValue` min value

`maxValue` max value

`...` any other arg

**Method** `validate()`**:** Validates data based on a range

*Usage:*

```
vrule_range$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_range$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_raw_codelist` *vrule_raw_codelist*

---

## Description

vrule_raw_codelist

vrule_raw_codelist

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_simple` -> `vrule_raw_codelist`

## Public fields

`ref_values` ref values

## Methods

### Public methods:

- vrule_raw_codelist$new()
- vrule_raw_codelist$validate()
- vrule_raw_codelist$clone()

**Method** `new()`**:** Initializes a raw codelist validation rule

*Usage:*

```
vrule_raw_codelist$new(ref_values = NULL, ...)
```

*Arguments:*

`ref_values` the raw codelists values

`...` any other arg

**Method** `validate()`**:** Validates value with a raw codelist validation rule

*Usage:*

```
vrule_raw_codelist$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_raw_codelist$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_report` *vrule_report*

---

## Description

vrule_report

vrule_report

## Public fields

`valid` valid

`report` report

## Methods

### Public methods:

- vrule_report$new()
- vrule_report$clone()

**Method** `new()`**:** Initializes a validation report

*Usage:*

```
vrule_report$new(valid = NULL, report = NULL)
```

*Arguments:*

`valid` `TRUE` or `FALSE`

`report` report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_report$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_threshold` *vrule_threshold*

---

## Description

vrule_threshold

vrule_threshold

## Details

vrule_threshold

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_operator_binary` -> `vrule::vrule_operator_relational` -> `vrule_threshold`

## Methods

### Public methods:

- vrule_threshold$new()
- vrule_threshold$validate()
- vrule_threshold$clone()

**Method** `new()`**:** Initializes a threshold validation rule

*Usage:*

```
vrule_threshold$new(operator, threshold, ...)
```

*Arguments:*

`operator` operator

`threshold` threshold

`...` any other arg

**Method** `validate()`**:** Validates data based on a threshold

*Usage:*

```
vrule_threshold$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_threshold$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`vrule_year` *vrule_year*

---

## Description

vrule_year

vrule_year

## Super classes

`vrule::vrule_abstract` -> `vrule::vrule_abstract_simple` -> `vrule::vrule_datatype` -> `vrule::vrule_integer` -> `vrule_year`

## Methods

### Public methods:

- `vrule_year$new()`
- `vrule_year$validate()`
- `vrule_year$clone()`

**Method** `new()`**:** Initializes a year validation rule

*Usage:*

```
vrule_year$new(na_allowed = FALSE, ...)
```

*Arguments:*

`na_allowed` `TRUE` if NA values are allowed, `FALSE` `otherwise`

`...` any other arg

**Method** `validate()`**:** Validates a year

*Usage:*

```
vrule_year$validate(value, ...)
```

*Arguments:*

`value` value

`...` any other args

*Returns:* a validation report, object of class vrule_report

**Method** `clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
vrule_year$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

# Index