

# Package ‘schematic’

May 9, 2026

**Title** Tidy Schema Validation for Data Frames

**Version** 0.1.2

**Description** Validate data.frames against schemas to ensure that data matches expectations. Define schemas using 'tidyselect' and predicate functions for type consistency, nullability, and more. Schema failure messages can be tailored for non-technical users and are ideal for user-facing applications such as in 'shiny' or 'plumber'.

**Maintainer** Will Hipson <will.e.hipson@gmail.com>

**License** MIT + file LICENSE

**URL** <https://github.com/whipson/schematic>,  
<https://whipson.github.io/schematic/>

**BugReports** <https://github.com/whipson/schematic/issues>

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Imports** cli, glue, purrr, rlang, tidyselect

**Suggests** knitr, rmarkdown, quarto, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Depends** R (>= 4.1.0)

**VignetteBuilder** knitr, quarto

**NeedsCompilation** no

**Author** Will Hipson [aut, cre, cph] (ORCID:  
<<https://orcid.org/0000-0002-3931-2189>>)

**Repository** CRAN

**Date/Publication** 2025-05-28 12:10:01 UTC

## Contents

check_schema . . . . .	2
is_all_distinct . . . . .	3
is_incrementing . . . . .	3

is_non_null . . . . .	4
is_positive_integer . . . . .	4
is_text . . . . .	5
is_whole_number . . . . .	5
last_check_errors . . . . .	6
mod_infiniteable . . . . .	7
mod_nullable . . . . .	7
print.Schema . . . . .	8
schema . . . . .	8

**Index** **10**

---

check_schema	<i>Validate a data.frame against a schema</i>
--------------	---

---

**Description**

Validate a data.frame against a schema

**Usage**

```
check_schema(data, schema)
```

**Arguments**

data	A data.frame to check
schema	A Schema object created with 'schema()'

**Value**

invisible if validation passes, otherwise stops with error

**Examples**

```
my_schema <- schema(
  mpg ~ is.numeric
)

check_schema(mtcars, my_schema)
```

---

is\_all\_distinct      *Check if all values in a vector are distinct*

---

**Description**

Check if all values in a vector are distinct

**Usage**

```
is_all_distinct(x)
```

**Arguments**

x                      A vector

**Value**

TRUE if the vector has all unique values

**Examples**

```
is_all_distinct(c(1:5)) # TRUE  
is_all_distinct(c(1, 1, 2)) # FALSE
```

---

is\_incrementing      *Check if the vector is sorted numerically or alphanumerically*

---

**Description**

'NA's are not ignored and any vector with 'NA's will fail unless the whole vector is 'NA'.

**Usage**

```
is_incrementing(x)
```

**Arguments**

x                      A vector

**Value**

TRUE if the vector is sorted

**Examples**

```
is_incrementing(1:5) # TRUE  
is_incrementing(letters[1:5]) # TRUE  
is_incrementing(c(4, 3, 0)) # FALSE
```

---

is_non_null	<i>Check if all values are not NA</i>
-------------	---------------------------------------

---

**Description**

Check if all values are not NA

**Usage**

```
is_non_null(x)
```

**Arguments**

x                    A vector

**Value**

TRUE if the vector has no NA values

**Examples**

```
is_non_null(1:5) # TRUE  
is_non_null(c(1, NA, 3)) # FALSE
```

---

is_positive_integer	<i>Check if a vector has all positive integers</i>
---------------------	--

---

**Description**

A positive integer is a whole number that is greater than 0.

**Usage**

```
is_positive_integer(x)
```

**Arguments**

x                    A vector

**Details**

This check requires 'is.integer(x)' to be true. If you want a more flexible check that allows for numbers of type 'numeric' but still want them to be integers, then use 'is\_whole\_number()'. 'NA's are ignored as long as they are 'NA\_integer'.

**Value**

TRUE if all elements are positive integers (NA ignored)

**Examples**

```
is_positive_integer(c(1L, 2L, 4L)) # TRUE
is_positive_integer(2.4) # FALSE
is_positive_integer(-3) # FALSE
```

---

is\_text

*Check if a vector is text-based (character or factor)*

---

**Description**

'NA's are ignored as long as they are 'NA\_character\_'.

**Usage**

```
is_text(x)
```

**Arguments**

x                    A vector

**Value**

TRUE if vector is either character or factor

**Examples**

```
is_text(letters[1:4]) # TRUE
is_text(as.factor(letters[1:4])) # TRUE
is_text(1) # FALSE
```

---

is\_whole\_number

*Check if a vector has all whole numbers*

---

**Description**

Similar to 'is\_positive\_integer()' but without the constraint that the underlying data type is actually integer. Useful if the numbers are stored as 'numeric' but you want to check that they are whole.

**Usage**

```
is_whole_number(x)
```

**Arguments**

x                    A vector

**Details**

'NA's are ignored.

**Value**

TRUE if all elements are whole numbers (NA ignored)

**Examples**

```
is_whole_number(c(2.0, 4.0)) # TRUE
is_whole_number(c(-1.4)) # FALSE
```

---

last_check_errors	<i>Retrieve latest schematic run time errors</i>
-------------------	--

---

**Description**

Predicates that error will store the error messages internally and these can be accessed here.

**Usage**

```
last_check_errors()
```

**Value**

error messages

**Examples**

```
last_check_errors()
```

---

mod_infinite	<i>Ignore infinite values in a predicate</i>
--------------	--

---

**Description**

This modifies a predicate function to ignore Inf.

**Usage**

```
mod_infinite(pred)
```

**Arguments**

pred            A predicate function

**Value**

A new predicate that ignores infinities

**Examples**

```
# The `is_incrementing` predicate will fail here
x <- c(1, Inf, 3)
is_incrementing(x) # FALSE

is_incrementing_inf <- mod_infinite(is_incrementing)
is_incrementing_inf(x) # TRUE
```

---

mod_nullable	<i>Allow NA in a predicate</i>
--------------	--------------------------------

---

**Description**

This modifies a predicate function to ignore NAs.

**Usage**

```
mod_nullable(pred)
```

**Arguments**

pred            A predicate function

**Value**

A new predicate that allows NAs

**Examples**

```
# The `is_incrementing` predicate will fail if there are NAs
x <- c(1, NA, 3)
is_incrementing(x) # FALSE

is_incrementing_null <- mod_nullable(is_incrementing)
is_incrementing_null(x) # TRUE
```

---

```
print.Schema          Print method for Schema
```

---

**Description**

Print method for Schema

**Usage**

```
## S3 method for class 'Schema'
print(x, ...)
```

**Arguments**

```
x          Object of class Schema
...        Other arguments passed to 'print()'
```

**Value**

invisible

---

```
schema          Create a schema object
```

---

**Description**

Create a schema object

**Usage**

```
schema(...)
```

**Arguments**

```
...          Formulae of the form tidyselct_expr ~ predicate
```

**Value**

A Schema object

**Examples**

```
# Simple schema with one declared column
my_schema <- schema(
  mpg ~ is.double
)

# Multiple columns
my_schema <- schema(
  Sepal.Length ~ is.numeric,
  Species ~ is.factor
)

# Use tidyselect syntax and anonymous functions
my_schema <- schema(
  starts_with("Sepal") ~ is.numeric,
  c(Petal.Length, Petal.Width) ~ function(x) all(x > 0)
)

# Use named arguments to customize error messages
my_schema <- schema(
  `Must be a positive number` = cyl ~ function(x) all(x > 0)
)
```

# Index

`check_schema`, 2

`is_all_distinct`, 3

`is_incrementing`, 3

`is_non_null`, 4

`is_positive_integer`, 4

`is_text`, 5

`is_whole_number`, 5

`last_check_errors`, 6

`mod_infiniteable`, 7

`mod_nullable`, 7

`print.Schema`, 8

`schema`, 8