Package 'S7schema'

October 26, 2026

Title 'S7' Framework for Schema-Validated YAML Configuration

Version 0.2.1

Description Provides a generic framework for working with YAML (YAML Ain't Markup Language) configuration files. Uses 'ajv' (Another JSON Schema Validator) via 'QuickJSR' to validate configurations against JSON Schema definitions. Configuration objects inherit from 'S7' classes and base lists, supporting downstream extension through custom classes and methods.

License Apache License (>= 2)

URL https://novonordisk-opensource.github.io/S7schema/, https://github.com/NovoNordisk-OpenSource/S7schema

BugReports https://github.com/NovoNordisk-OpenSource/S7schema/issues

Depends R (>= 4.1)

Imports cli, jsonlite, QuickJSR (>= 1.3.1), rlang, S7, tools, yaml (>= 2.3.8)

Suggests knitr, purrr, rmarkdown, testthat (>= 3.0.0), tibble, tidyr, withr

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

Author Aksel Thomsen [aut, cre],
Matthew Phelps [aut],
Novo Nordisk A/S [cph],
Evgeny Poberezkin [cph] (Author of included ajv, fast-deep-equal, and json-schema-traverse libraries),
Vincent Le Goff [cph] (Author of included fast-uri library)

Maintainer Aksel Thomsen <oath@novonordisk.com>

Repository CRAN

Date/Publication 2026-10-26 16:10:02 UTC

Contents

document_schema 2
S7schema 3
to_yaml 4
validate_config 6
write_config 8

Index **9**

`document_schema` *Document configuration schema*

Description

Creates markdown documentation of a configuration suitable for use in e.g. vignettes to provide easy readable documentation for users.

Usage

```
document_schema(x, header_start_level = 1L)
```

Arguments

`x`	`character(1)` path to JSON schema, `list()` of already loaded specifications, or `S7schema()` object.
`header_start_level`	`numeric(1)` Level of initial header. All subheaders will continuously be one level smaller.

Value

`character(1)`/`knitr::asis_output()` markdown with the documentation.

Examples

```
# Simple example schema
system.file("examples/schema.json", package = "S7schema") |>
  document_schema(2) |>
  cat()

# Changing header start level to 1
system.file("examples/schema.json", package = "S7schema") |>
  document_schema(1) |>
  cat()

# Example with definitions
system.file("examples/definitions.json", package = "S7schema") |>
  document_schema(2) |>
  cat()
```

S7schema *Work with valid configurations*

Description

`S7schema()` provides a generic way of working with yaml configuration files.

The object is created by supplying both an initial YAML configuration (`file`) and the JSON schema definition (`schema`) of the configuration file.

The initial configuration is validated before the new object is returned. If not valid the first error is thrown, together with a path to the entry in the YAML file, and a description of the error.

The `S7schema` class inherits from `list`, ensuring that the content of the YAML file can be accessed as if read directly with `yaml::read_yaml()`, and supports the below workflow:

1. Read and validate config file: `x <- S7schema(...)`
2. Edit content as if it was a list: `x$new_entry <- "new_value"`
3. Validate new content against the original schema: `validate(x)`
4. Use values in downstream functions: `x$new_entry`

Usage

```
S7schema(file, schema, .data)
```

Arguments

`file`	`character(1)` path to a yaml file to be checked. Or `NULL` if created using `.data`.
`schema`	`character(1)` path to a JSON schema.
`.data`	`list()` input to create the object. Mutually exclusive with `file`.

Details

The content is validated as R reads it: files are read with `yaml::read_yaml()`, and `S7schema` objects are converted with `to_yaml()` and read back before validation. `list` objects in `validate_list()` are validated as they are. See internal `validator()` documentation for more info on how the validation is done.

In YAML files and `S7schema` objects, R reads a YAML sequence with one item (`[a]`) as a single value (`"a"`). For fields that can hold one or more values, allow both forms in the schema with `oneOf`:

```
{
  "oneOf": [
    {"type": "string"},
    {"type": "array", "items": {"type": "string"}}
  ]
}
```

NA, NaN, and Inf values are validated as null. A field that may be missing must allow null in the schema:

```
{"type": ["string", "null"]}
```

Numbers with up to 16 significant digits are validated exactly. Numbers that need 17, e.g. 0.1 + 0.2, are validated exactly in files and lists, but rounded in S7schema objects, as to_yaml() writes them.

Value

New S7schema object.

Properties

schema character(1) path to JSON schema being used to validate against.

validator Internal validator() used to validate the content (read-only).

file character(1) path to the source YAML file (or NULL if not set).

Examples

```
# Work with yaml configuration file:
S7schema(
  file = system.file("examples/config.yml", package = "S7schema"),
  schema = system.file("examples/schema.json", package = "S7schema")
)

# Create object in memory
S7schema(
  .data = list(my_config_var = 6),
  schema = system.file("examples/schema.json", package = "S7schema")
)
```

---

to_yaml *Convert an R object to YAML*

Description

This function is used internally when validating S7schema objects, and when using write_config() to save a configuration.

Underneath it is calling yaml::as.yaml() to do the conversion, but all logical values are converted to true/false instead of yes/no respectively for a more robust integration with other YAML parsers. Date values are written as ISO 8601 strings ("2025-08-06"), and numbers are written with precision = 15 instead of the default of 7. Numbers with up to 16 significant digits are read back as the same value, while numbers that need 17, e.g. 0.1 + 0.2, are rounded.

Missing values are written as:

- logical NA as .na
- NA_character_ as .na.character
- NA_integer_ as .na.integer
- NA_real_ as .na.real

R reads all of them back as NA, while other YAML parsers read them as strings. Previously logical NA was written as NA, which R read back as the string "NA".

It is rarely relevant to call this function directly except for debugging purposes, or when implementing a new method for your own object class.

Usage

```
to_yaml(x)
```

Arguments

x	object to convert to YAML.

Details

to_yaml() dispatches based on the class of x. Register a new S7 method if you want to overwrite how your own class is converted to YAML. See S7::method() for more information.

The default method uses yaml::verbatim_logical() to overwrite the default behavior of handling logical values, and a handler for Date values. The logical handler writes NA as .na, so it is read back as NA:

```
function(x) {
  yaml::as.yaml(
    x = x,
    precision = 15,
    handlers = list(
      logical = \(x) {
        x <- yaml::verbatim_logical(x)
        x[is.na(x)] <- ".na"
        x
      },
      Date = \(x) format(x, "%Y-%m-%d")
    )
  )
}
```

Copy this and add your own additional handlers when implementing a new method.

Value

character(1) YAML string.

Examples

```
# Convert simple list to YAML
to_yaml(list(hello = "world", is_today = TRUE)) |>
  cat()

# Missing logical values are written as .na
to_yaml(list(a = NA, b = c(TRUE, NA))) |>
  cat()

# Convert S7schema object
x <- S7schema(
  file = system.file("examples/config.yml", package = "S7schema"),
  schema = system.file("examples/schema.json", package = "S7schema")
)

print(x)

to_yaml(x) |>
  cat()
```

---

validate_config *One-shot validation of configurations*

Description

Check if a configuration is in accordance with a JSON schema definition.

It is possible to either validate an existing `list` object in memory or an existing `yaml` configuration file.

Usage

```
validate_list(x, schema)

validate_yaml(file, schema)
```

Arguments

`x`	`list` object to validate
`schema`	`character(1)` path to a JSON schema.
`file`	`character(1)` path to a yaml file to be checked. Or `NULL` if created using `.data`.

Details

The content is validated as R reads it: files are read with `yaml::read_yaml()`, and `S7schema` objects are converted with `to_yaml()` and read back before validation. `list` objects in `validate_list()` are validated as they are. See internal `validator()` documentation for more info on how the validation is done.

In YAML files and S7schema objects, R reads a YAML sequence with one item ([a]) as a single value ("a"). For fields that can hold one or more values, allow both forms in the schema with oneOf:

```
{
  "oneOf": [
    {"type": "string"},
    {"type": "array", "items": {"type": "string"}}
  ]
}
```

NA, NaN, and Inf values are validated as null. A field that may be missing must allow null in the schema:

```
{"type": ["string", "null"]}
```

Numbers with up to 16 significant digits are validated exactly. Numbers that need 17, e.g. 0.1 + 0.2, are validated exactly in files and lists, but rounded in S7schema objects, as to_yaml() writes them.

Value

- validate_list(): invisible(x)
- validate_yaml(): invisible(file)

See Also

S7schema()

Examples

```
# Validate list object in memory
validate_list(
  x = list(my_config_var = 1),
  schema = system.file("examples/schema.json", package = "S7schema")
) |>
  print()

# Validate yaml file on disk
validate_yaml(
  file = system.file("examples/config.yml", package = "S7schema"),
  schema = system.file("examples/schema.json", package = "S7schema")
) |>
  print()
```

write_config *Write YAML configuration file*

Description

Thin wrapper around `to_yaml()` calling `validate()` before converting to YAML and creating the file, ensuring that the saved configuration is valid.

Usage

```
write_config(x, path = NULL)
```

Arguments

`x`	`S7schema` object to write.
`path`	`character(1)` path to the file to write to. Default `NULL` uses `x@file` for `S7schema()` objects.

Value

Invisible `x` (the input `S7schema` object). Called for side effect of writing the file.

Examples

```
# Read configuration file:
x <- S7schema(
  file = system.file("examples/config.yml", package = "S7schema"),
  schema = system.file("examples/schema.json", package = "S7schema")
)

print(x)

# Edit content
x$my_config_var <- 2

# Save new file
write_config(
  x = x,
  path = tempfile(fileext = ".yml")
)
```

Index

document_schema, 2

S7schema, 3
S7schema(), *7*

to_yaml, 4

validate_config, 6
validate_list *(*validate_config*)*, 6
validate_yaml *(*validate_config*)*, 6
validator(), *3*, *4*, 6

write_config, 8