Package: corvusjsonschema (via r-universe)

October 17, 2026

Title A High-Performance 'JSON Schema' Validator

Version 0.1.2

Description Validates R values and 'JSON' text against 'JSON Schema' drafts 4, 6, 7, 2019-09 and 2020-12, with results collection and annotations. A schema is compiled once into a validator, which reads R lists and vectors in place, or parses 'JSON' text itself. The evaluator is the 'corvus-json-schema' 'Rust' crate, which passes the whole 'JSON-Schema-Test-Suite'.

License Apache License (>= 2)

Copyright The installed package holds Unicode data and third-party code compiled from Rust crates. See the file inst/THIRD-PARTY-NOTICES.txt for their notices.

URL https://github.com/corvus-dotnet/Corvus.JsonSchema

BugReports https://github.com/corvus-dotnet/Corvus.JsonSchema/issues

Depends R (>= 4.2)

Suggests jsonlite

SystemRequirements Cargo (Rust's package manager), rustc (>= 1.89)

Encoding UTF-8

NeedsCompilation yes

Config/pak/sysreqs libclang-dev

Repository https://corvus-dotnet.r-universe.dev

Date/Publication 2026-10-17 10:42:49 UTC

RemoteUrl https://github.com/corvus-dotnet/Corvus.JsonSchema

RemoteRef r-release

RemoteSha 5cce06fd06075b80598895b129751ba78a43ba53

RemoteSubdir src-r/corvusjsonschema

Contents

corvusjsonschema-package 2
compile_schema 2
crate_version 4
evaluate 4
is_valid 6

Index **8**

`corvusjsonschema-package` *A High-Performance JSON Schema Validator*

Description

Validates R values and JSON text against JSON Schema drafts 4, 6, 7, 2019-09 and 2020-12, with results collection and annotations.

Compile a schema once with `compile_schema`. Then `is_valid` validates an R value, read in place, and `is_valid_json` validates JSON text. `evaluate` and `evaluate_json` report why a value is not valid, and the annotations a schema makes.

The evaluator is the `corvus-json-schema` Rust crate, one of a family of implementations of one design in several languages. It passes every test of the JSON-Schema-Test-Suite.

Author(s)

Matthew Adams, Endjin Limited.

See Also

https://github.com/corvus-dotnet/Corvus.JsonSchema

`compile_schema` *Compile a JSON Schema*

Description

Compiles a JSON Schema (draft 4, 6, 7, 2019-09 or 2020-12) into a validator. Compile a schema once and use the validator for every value: compiling does the work that makes validating fast.

Usage

```
compile_schema(schema, default_dialect = "draft2020-12", assert_format = NA,
               assert_format_in_legacy_drafts = FALSE, assert_content = TRUE,
               formats = NULL, resolver = NULL, base_uri = NULL,
               entry_point = NULL, max_depth = 128L, pattern_match_limit = 1)
```

Arguments

`schema` The schema: one string of JSON text, or R values (a list, or `TRUE` or `FALSE`) read as `is_valid` reads a value.

`default_dialect` The dialect of a schema that has no `$schema`: `"draft4"`, `"draft6"`, `"draft7"`, `"draft2019-09"` or `"draft2020-12"`.

`assert_format` Whether `format` is asserted. `NA` follows the dialect's vocabularies, under which `format` is an annotation unless the schema's metaschema says otherwise.

`assert_format_in_legacy_drafts` With `assert_format = NA`, whether `format` is asserted in draft 4 to 7.

`assert_content` Whether `contentEncoding` and `contentMediaType` are asserted in draft 7, the only draft that asserts them.

`formats` A named list of functions, one for each custom format. Each takes the string and returns whether it is valid. They take precedence over the built-in formats.

`resolver` A function that gives the document of an absolute URI the schema refers to: JSON text, R values, or `NULL` when it does not know the document. The standard metaschemas are always known.

`base_uri` The base URI of the schema, for a schema with relative references and no `$id`.

`entry_point` A reference, relative to the schema's root, to the subschema to validate against, such as `"#/$defs/item"`.

`max_depth` How many times a schema may refer to itself without moving through the value before evaluation is abandoned with an error.

`pattern_match_limit` The seconds one match of a `pattern` or `patternProperties` regular expression against one string may take. A match that takes longer is abandoned with an error, and is never an answer. `NULL`, zero or less turns the limit off, for a schema and instances that are both trusted.

Value

An object of class `corvus_json_schema`, for `is_valid`, `is_valid_json`, `evaluate` and `evaluate_json`. It holds compiled native code, so it does not survive `saveRDS` or a saved workspace: compile the schema again in a new session.

Conditions

A schema that cannot be compiled raises a condition of class `corvus_compilation_error`, and one with a regular expression over a limit of size (longer than 1,048,576 bytes, nested more than 1,000 groups deep, or of more than about 16,000 terms) raises `corvus_pattern_too_large_error`, which is a `corvus_compilation_error`. A match of a `patternProperties` regular expression against a property name the schema declares can raise `corvus_pattern_match_limit_error`. Schema text that is not JSON, `corvus_invalid_json_error`; and an R value that is not a JSON value, `corvus_value_error`. All inherit from `corvus_json_schema_error`.

Examples

```
validator <- compile_schema('{
  "type": "object",
  "required": ["id"],
  "properties": {"id": {"type": "integer", "minimum": 1}}
}')
is_valid(validator, list(id = 3))
is_valid_json(validator, '{"id": 0}')

# The same schema as R values.
compile_schema(list(type = "object", required = list("id")))

# A custom format.
even <- compile_schema('{"format": "even-length"}', assert_format = TRUE,
                       formats = list("even-length" = function(s) nchar(s) %% 2 == 0))
is_valid(even, "ab")
```

---

crate_version *The Version of the Evaluator*

Description

The version of the `corvus-json-schema` Rust crate the package was built from.

Usage

```
crate_version()
```

Value

One string, such as `"0.1.7"`.

Examples

```
crate_version()
```

---

evaluate *Evaluate a Value and Collect the Results*

Description

Evaluates an R value (`evaluate`) or JSON text (`evaluate_json`) against a compiled schema without stopping at the first failure, and returns what each keyword found.

Usage

```
evaluate(validator, value, level = "detailed")

evaluate_json(validator, json, level = "detailed")
```

Arguments

validator	A compiled schema, from compile_schema.
value	An R value, read as is_valid reads one.
json	One string of JSON text.
level	How much to report: "basic" (the failures, without messages), "detailed" (the failures, with messages) or "verbose" (every keyword evaluated, passing ones included, and the annotations).

Value

An object of class corvus_evaluation, a list with:

valid	Whether the value is valid.
level	The level asked for.
results	A data frame with a row for each result, the root's own among them: valid, message, evaluation_location (the path of keywords followed, through references), schema_location (where the keyword is in the schema) and instance_location (where the value is in the instance), the three locations as JSON Pointers.
annotations	At the "verbose" level, a data frame with a row for each annotation: instance_location, keyword, schema_location and value (a list column of the values, as R values). NULL at the other levels.

Examples

```
validator <- compile_schema('{
  "type": "object",
  "required": ["name"],
  "properties": {"name": {"type": "string", "title": "Name"},
                 "age": {"type": "integer", "minimum": 0}}
}')
result <- evaluate(validator, list(age = -1))
result
result$results[, c("instance_location", "message")]

evaluate_json(validator, '{"name": "Ada"}', level = "verbose")$annotations
```

is_valid — *Validate a Value or JSON Text*

Description

`is_valid` says whether an R value is valid against a compiled schema, reading the value in place. `is_valid_json` says whether JSON text is, parsing the text itself, for each element of a character vector. Both stop at the first failure. Use `evaluate` to find out why a value is not valid.

Usage

```
is_valid(validator, value)

is_valid_json(validator, json)
```

Arguments

`validator`	A compiled schema, from `compile_schema`.
`value`	An R value, read as JSON as described below.
`json`	A character vector of JSON texts.

Value

`is_valid` returns `TRUE` or `FALSE`. `is_valid_json` returns a logical vector as long as `json`, with `NA` for an `NA` element.

How R values read as JSON

- `NULL` is `null`.
- A list with names is an object, and a list without names is an array. So `list()` is `[]`, and an empty named list is `{}`. This is how the **jsonlite** package's `fromJSON(simplifyVector = FALSE)` gives JSON values. Where two elements have one name, the first is read.
- A logical, integer, double or character vector of length one is a scalar. A vector of any other length is an array of scalars, and its names are ignored. Wrap a vector of length one in `I` for it to read as an array of one element.
- `NA` of any type is `null`.
- A whole double, such as `3`, is an integer as well as a number, in every dialect.
- Strings are read as UTF-8 whatever encoding they are marked with.

Nothing else is a JSON value: not `NaN` or an infinity, a function, an environment, a raw or complex vector, or an object with a class, such as a data frame, a factor or a date. Convert such a value first. A value is read only where the schema examines it, so a value the schema ignores is never an error.

An R string cannot hold a NUL character, so JSON text with one in a string can only be validated as text.

Conditions

A value that is not a JSON value raises `corvus_value_error`; text that is not JSON, `corvus_invalid_json_error`; a schema that refers to itself more than `max_depth` times without moving through the value, `corvus_depth_error`; a match of a regular expression that takes longer than the `pattern_match_limit` of `compile_schema`, `corvus_pattern_match_limit_error`, for which the value is neither valid nor not valid; and an error in a custom format function, `corvus_callback_error`. All inherit from `corvus_json_schema_error`.

Examples

```
validator <- compile_schema('{"type": "array", "items": {"type": "integer"}}')
is_valid(validator, 1:3)
is_valid(validator, list(1, 2.5))
is_valid(validator, 1)        # a scalar, not an array
is_valid(validator, I(1))     # an array of one
is_valid_json(validator, c("[1, 2]", '["a"]', NA))
```

Index

∗ **package**
    corvusjsonschema-package, 2

compile_schema, 2, *2*, *5–7*
corvusjsonschema
        *(*corvusjsonschema-package*)*, 2
corvusjsonschema-package, 2
crate_version, 4

evaluate, *2, 3*, 4, *6*
evaluate_json, *2, 3*
evaluate_json *(*evaluate*)*, 4

I, *6*
is_valid, *2, 3*, *5*, 6
is_valid_json, *2, 3*
is_valid_json *(*is_valid*)*, 6

print.corvus_evaluation *(*evaluate*)*, 4
print.corvus_json_schema
        *(*compile_schema*)*, 2

saveRDS, *3*