Package 'jsonvalidate'

October 13, 2022

Title Validate 'JSON' Schema

Version 1.3.2

Maintainer Rich FitzJohn <rich.fitzjohn@gmail.com>

Description Uses the node library 'is-my-json-valid' or 'ajv' to validate 'JSON' against a 'JSON' schema. Drafts 04, 06 and 07 of 'JSON' schema are supported.

License MIT + file LICENSE

URL https://docs.ropensci.org/jsonvalidate/, https://github.com/ropensci/jsonvalidate

BugReports https://github.com/ropensci/jsonvalidate/issues

Imports V8

Suggests knitr, jsonlite, rmarkdown, testthat, withr

RoxygenNote 7.1.2

VignetteBuilder knitr

Encoding UTF-8

Language en-GB

Config/testthat/edition 3

NeedsCompilation no

Author Rich FitzJohn [aut, cre],
Rob Ashton [aut],
Alex Hill [ctb],
Alicia Schep [ctb],
Ian Lyttle [ctb],
Kara Woo [ctb],
Mathias Buus [aut, cph] (Author of bundled imjv library),
Evgeny Poberezkin [aut, cph] (Author of bundled Ajv library)

Repository CRAN

Date/Publication 2021-11-03 15:50:02 UTC

R topics documented:

`json_validate` *Validate a json file*

Description

Validate a single json against a schema. This is a convenience wrapper around `json_validator(schema)(json)`. See `json_validator()` for further details.

Usage

```
json_validate(
  json,
  schema,
  verbose = FALSE,
  greedy = FALSE,
  error = FALSE,
  engine = "imjv",
  reference = NULL,
  query = NULL,
  strict = FALSE
)
```

Arguments

`json`	Contents of a json object, or a filename containing one.
`schema`	Contents of the json schema, or a filename containing a schema.
`verbose`	Be verbose? If `TRUE`, then an attribute "errors" will list validation failures as a data.frame
`greedy`	Continue after the first error?
`error`	Throw an error on parse failure? If `TRUE`, then the function returns `NULL` on success (i.e., call only for the side-effect of an error on failure, like `stopifnot`).
`engine`	Specify the validation engine to use. Options are "imjv" (the default; which uses "is-my-json-valid") and "ajv" (Another JSON Schema Validator). The latter supports more recent json schema features.
`reference`	Reference within schema to use for validating against a sub-schema instead of the full schema passed in. For example if the schema has a 'definitions' list including a definition for a 'Hello' object, one could pass "#/definitions/Hello" and the validator would check that the json is a valid "Hello" object. Only available if `engine = "ajv"`.

query — A string indicating a component of the data to validate the schema against. Eventually this may support full jsonpath syntax, but for now this must be the name of an element within `json`. See the examples for more details.

strict — Set whether the schema should be parsed strictly or not. If in strict mode schemas will error to "prevent any unexpected behaviours or silently ignored mistakes in user schema". For example it will error if encounters unknown formats or unknown keywords. See https://ajv.js.org/strict-mode.html for details. Only available in `engine = "ajv"`.

Examples

```
# A simple schema example:
schema <- '{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Product",
    "description": "A product from Acme\'s catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
        "name": {
            "description": "Name of the product",
            "type": "string"
        },
        "price": {
            "type": "number",
            "minimum": 0,
            "exclusiveMinimum": true
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
            },
            "minItems": 1,
            "uniqueItems": true
        }
    },
    "required": ["id", "name", "price"]
}'

# Test if some (invalid) json conforms to the schema
jsonvalidate::json_validate("{}", schema, verbose = TRUE)

# Test if some (valid) json conforms to the schema
json <- '{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
```

```
}'
jsonvalidate::json_validate(json, schema)

# Test a fraction of a data against a reference into the schema:
jsonvalidate::json_validate(json, schema,
                            query = "tags", reference = "#/properties/tags",
                            engine = "ajv", verbose = TRUE)
```

`json_validator` *Create a json validator*

Description

Create a validator that can validate multiple json files.

Usage

```
json_validator(schema, engine = "imjv", reference = NULL, strict = FALSE)
```

Arguments

`schema`	Contents of the json schema, or a filename containing a schema.
`engine`	Specify the validation engine to use. Options are "imjv" (the default; which uses "is-my-json-valid") and "ajv" (Another JSON Schema Validator). The latter supports more recent json schema features.
`reference`	Reference within schema to use for validating against a sub-schema instead of the full schema passed in. For example if the schema has a 'definitions' list including a definition for a 'Hello' object, one could pass "#/definitions/Hello" and the validator would check that the json is a valid "Hello" object. Only available if `engine = "ajv"`.
`strict`	Set whether the schema should be parsed strictly or not. If in strict mode schemas will error to "prevent any unexpected behaviours or silently ignored mistakes in user schema". For example it will error if encounters unknown formats or unknown keywords. See https://ajv.js.org/strict-mode.html for details. Only available in `engine = "ajv"`.

Validation Engines

We support two different json validation engines, `imjv` ("is-my-json-valid") and `ajv` ("Another JSON Validator"). `imjv` was the original validator included in the package and remains the default for reasons of backward compatibility. However, users are encouraged to migrate to `ajv` as with it we support many more features, including nested schemas that span multiple files, meta schema versions later than draft-04, validating using a subschema, and validating a subset of an input data object.

If your schema uses these features we will print a message to screen indicating that you should update when running interactively. We do not use a warning here as this will be disruptive to users. You

can disable the message by setting the option `jsonvalidate.no_note_imjv` to `TRUE`. Consider using `withr::with_options()` (or simply `suppressMessages()`) to scope this option if you want to quieten it within code you do not control. Alternatively, setting the option `jsonvalidate.no_note_imjv` to `FALSE` will print the message even noninteractively.

Updating the engine should be simply a case of adding `{engine = "ajv"` to your `json_validator` or `json_validate` calls, but you may see some issues when doing so.

- Your json now fails validation: We've seen this where schemas spanned several files and are silently ignored. By including these, your data may now fail validation and you will need to either fix the data or the schema.
- Your code depended on the exact payload returned by `imjv`: If you are inspecting the error result and checking numbers of errors, or even the columns used to describe the errors, you will likely need to update your code to accommodate the slightly different format of `ajv`
- Your schema is simply invalid: If you reference an invalid metaschema for example, jsonvalidate will fail

Using multiple files

Multiple files are supported. You can have a schema that references a file `child.json` using `{"$ref": "child.json"}` - in this case if `child.json` includes an `id` or `$id` element it will be silently dropped and the filename used to reference the schema will be used as the schema id.

The support is currently quite limited - it will not (yet) read sub-child schemas relative to child schema `$id` url, and does not support reading from URLs (only local files are supported).

Examples

```
# A simple schema example:
schema <- '{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Product",
    "description": "A product from Acme\'s catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
        "name": {
            "description": "Name of the product",
            "type": "string"
        },
        "price": {
            "type": "number",
            "minimum": 0,
            "exclusiveMinimum": true
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
```

```
            },
            "minItems": 1,
            "uniqueItems": true
        }
    },
    "required": ["id", "name", "price"]
}'

# Create a validator function
v <- jsonvalidate::json_validator(schema)

# Test if some (invalid) json conforms to the schema
v("{}", verbose = TRUE)

# Test if some (valid) json conforms to the schema
v('{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}')

# Using features from draft-06 or draft-07 requires the ajv engine:
schema <- "{
  '$schema': 'http://json-schema.org/draft-06/schema#',
  'type': 'object',
  'properties': {
    'a': {
      'const': 'foo'
    }
  }
}"

# Create the validator
v <- jsonvalidate::json_validator(schema, engine = "ajv")

# This confirms to the schema
v('{"a": "foo"}')

# But this does not
v('{"a": "bar"}')
```

Index