YAML Schema Standard

Release 0.1.0.dev0

Michael Droettboom, Erik Bray, et al
Space Telescope Science Institute

August 20, 2015

CONTENTS

1 Changes **3**
1.1 Version 0.1.0 3

2 Introduction **5**

3 History **7**

4 New Keywords **9**
4.1 `tag` keyword 9
4.2 `propertyOrder` keyword 10
4.3 `style` keyword 10
4.4 `flowStyle` keyword 10
4.5 `examples` keyword 11

5 Additional Notes **13**
5.1 Anchors/Aliases and References 13

6 Schemas **15**
6.1 draft-01: YAML Schema 15
6.2 invoice: Invoice 18

This document describes the YAML Schema format, an extension to JSON Schema designed specifically for validating and documenting data products serialized as YAML.

This document is a work in progress and does not represent a released version of the YAML Schema standard.

CHAPTER 1

CHANGES

1.1 Version 0.1.0

First pre-release.

CHAPTER 2

INTRODUCTION

YAML Schema (page 15) is a small extension to JSON Schema Draft 4 (http://json-schema.org/latest/json-schema-validation.html) so support features unique to YAML data serialization language (as opposed to plain JSON), as well as enhance documentation of schemas. Understanding JSON Schema (http://spacetelescope.github.io/understanding-json-schema/) provides a good resource for understanding how to use JSON Schema, and further resources are available at json-schema.org (http://json-schema.org). A working understanding of YAML and JSON Schema is assumed for this document, which only describes what makes YAML Schema different from JSON Schema.

A YAML schema as defined by this document is typically serialized as YAML, though may also be serialized as JSON (JSON being a subset of YAML), or any other format that can encapsulate the structure of JSON data. In fact, many JSON Schema validators work by deserializing a JSON document into native data structures of the language in which it is implemented, and checking that data structure against the schema. So although the schema itself is defined in JSON, JSON Schema is not strictly limited to data that has been at one time serialized as JSON. It is, however, limited to data structures that can be serialized as JSON.

Given some data structure, it will generally be possible to serialize it either as JSON or as YAML. However, YAML serializations may contain additional explicit structure that is not possible in JSON without use of local, application-specific conventions. Examples include tags, and ordered objects, both of which are described in more detail below. As YAML is designed with human-readability in mind, presentation is also of more concern, and the YAML specification has more to say on the topic than JSON. YAML often provides multiple options for how the same data can be presented (beyond just placement of whitespace), and a schema can be used to provide hints to YAML writers for how a given data structure should be serialized for optimal readability.

To be clear, the JSON Schema specification allows extensions by design [1], through definition of additional keywords that may be used in a schema. JSON Schema implementations that do not support the additional keywords should ignore them; as such, to support YAML Schema it is necessary to provide JSON Schema implementations that interpret the added keywords. Also, just as JSON Schema provides a metaschema [2], YAML Schema has a metaschema describing how to correctly interpret its additional keywords. The YAML Schema metaschema *extends* JSON Schema's metaschema using the JSON Schema extension capability, and as such is a superset of JSON Schema's metaschema.

[1] Extending the JSON Schema core definition (http://json-schema.org/latest/json-schema-core.html#anchor20)

[2] JSON Schema Meta Core Meta-Schema (https://github.com/Julian/jsonschema/blob/4baff2178f4ade789cb6049f2b6bcd9031c8f89f/jsonschema/schemas/draft
(on GitHub for ease of viewing)

CHAPTER 3

HISTORY

YAML Schema was originally developed in parallel with the specification and implementation of the ASDF file format (http://asdf-standard.readthedocs.org/en/latest/index.html), a new file format being developed at STScI (http://www.stsci.edu/portal/) for serializing astronomy and other scientific data.

It was an early requirement to include a validation mechanism for the core data structures appearing in ASDF files, and a strong desire to build this mechanism on existing, broadly adopted standards. JSON Schema quickly emerged as the best choice. However, ASDF serializes its core data structures as YAML (a superset of JSON, as of v1.2 of the YAML standard), and makes extensive use of YAML-specific features (chiefly tags). So it became desireable to extend JSON Schema to support validation of some YAML-specific featuers.

Additionally, though not particular to YAML, there was a desire to include more documentation for schemas within the schemas themselves. Although YAML (but notably not JSON) has support for in-line comments, those comments are ignored by parsers and are not returned as part of the data structure read out of a YAML file. It was advantageous to have documentation as part of the data structure for schemas themselves, as it allows better introspection of schemas either as part of a user API, or for generation of human-readable documentation. To that end YAML Schema adds additional documentation-related properties to the schema format. However, as these properties are not YAML-specific they could, in principle, be added as a separate JSON Schema extension.

Although YAML Schema was created specifically for ASDF, we expect it to have broader applicability, and hope that offering it as a separate product will encourage adoption of this format within the YAML community, and drive development of implementations.

CHAPTER 4

NEW KEYWORDS

YAML Schema adds five new keywords to JSON Schema:

- `tag` *keyword* (page 9)
- `propertyOrder` *keyword* (page 10)
- `style` *keyword* (page 10)
- `flowStyle` *keyword* (page 10)
- `examples` *keyword* (page 11)

4.1 tag keyword

`tag`, which may be attached to any data type, declares that the element must have the given YAML tag.

For example, the invoice schema declares its tag to be:

```
tag: "tag:stsci.edu:yaml-schema/examples/invoice"
```

This implies that an object in a YAML document is only matched to this schema if it explicitly marked with the `!invoice` tag. Conversely, if a YAML document references this schema, and objects that have the `!invoice` tag must satisify the schema associated with it. Therefore, there is a one-to-one mapping between YAML tags and schemas which specify that tag in the `tag` keyword.

A schema may require an individual object property or array item to have a specific tag by referencing the *schema* associated with that tag, rather than the tag directly. For example a schema that includes an invoice might look like:

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/yaml-schema/examples/customer"
tag: "tag:stsci.edu:yaml-schema/examples/customer"
title: Customer
properties:
    order-history:
        type: array
        items:
            $ref: "invoice"
```

In this example the reference to `"invoice"` as the `order-history` item type does not directly refer to the `invoice` *tag*, but to the invoice *schema*. There is no requirement for a schema referenced in this way to have an associated tag. But because the `"invoice"` schema does use the `tag:` keyword this has the net effect of requiring all `order-history` items to be tagged `!<tag:stsci.edu:yaml-schema/examples/invoice>` in the YAML document.

4.2 `propertyOrder` keyword

`propertyOrder`, which applies only to objects, declares that the object must have its properties presented in the given order.

TBD: It is not yet clear whether this keyword is necessary or desirable.

YAML provides an `!!omap` type [1] for *ordered* mappings. In JSON terms, this equates to semantically meaningful order to the properties in the object, which is not normally possible in JSON without a local convention. As native language support for ordered mappings is not implemented in all YAML parsers, applications may choose to ignore this keyword for validation purposes. However, this keyword may also be used as a presentation hint, informing the YAML writer on the order to write out keywords in a specific mapping object, where possible.

4.3 `style` keyword

Must be `inline`, `literal` or `folded`.

Specifies the default serialization style to use for a string. YAML supports multiple styles for strings:

```
Inline style: "First line\nSecond line"

Literal style: |
  First line
  Second line

Folded style: >
  First
  line

  Second
  line
```

This property gives an optional hint to the tool outputting the YAML which style to use. If not provided, the library is free to use whatever heuristics it wishes to determine the output style. This property does not enforce any particular style on YAML being parsed.

4.4 `flowStyle` keyword

Must be either `block` or `flow`.

Specifies the default serialization style to use for an array or object. YAML supports multiple styles for arrays/sequences and objects/maps, called "block style" and "flow style". For example:

```
Block style: !!map
 Clark : Evans
 Ingy  : döt Net
 Oren  : Ben-Kiki

Flow style: !!map { Clark: Evans, Ingy: döt Net, Oren: Ben-Kiki }
```

This property gives an optional hint to the tool outputting the YAML which style to use. If not provided, the library is free to use whatever heuristics it wishes to determine the output style. This property does not enforce any particular style on YAML being parsed.

[1] Ordered Mapping (omap) Type for YAML (http://yaml.org/type/omap.html)

4.5 `examples` keyword

The schema may contain a list of examples demonstrating how to use the schema. It is a list where each item is a pair. The first item in the pair is a prose description of the example, and the second item is YAML content (as a string) containing the example.

For example:

```
examples:
  -
    - Complex number: 1 real, -1 imaginary
    - "!complex 1-1j"
```

This keyword is purely for informational purposes, and while the examples may contain YAML, there is otherwise nothing YAML-specific about it. This keyword can help in generation of nice documentation for schema, as well as writing automated tests of the schema in-line with the schema definition itself.

CHAPTER 5

ADDITIONAL NOTES

5.1 Anchors/Aliases and References

Another feature of YAML that is not reflected in JSON is anchors and aliases– these allow an object that appears multiple times in the document to be written out just once along with an *anchor*. This object can than be referenced more than once via an *alias* node.

As this is mostly a presentation detail YAML Schema does not currently have anything to say about it. In principle YAML Schema could include hints for whether or not an object an object may use anchors or how those anchors should be named. However in practice we have yet to identify a need for this.

YAML schemas themselves *may* use anchors and aliases within the schema; however, this usage is discouraged. In practice we have found the JSON Pointer (http://tools.ietf.org/html/draft-pbryan-zyp-json-pointer-02) [1] syntax more useful for references within a schema. This is in part because it is already used in JSON Schema [2] via the JSON Reference (http://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03) standard, and because it enables both intra-schema references and references to external schemas (whereas YAML aliases only allow intra-document references). The support for external schema references is especially useful for extending and encapsulating existing schemas from disparate projects.

[1] For a softer introduction to how JSON Pointer is used, see the relevant section in Understanding JSON Schema (http://spacetelescope.github.io/understanding-json-schema/structuring.html#reuse)

[2] URL dereferencing in JSON Schema (http://json-schema.org/latest/json-schema-core.html#anchor25)

CHAPTER 6

SCHEMAS

This reference section includes the YAML Schema meta-schema and any example schemas.

6.1 draft-01: YAML Schema

Type: schema (http://json-schema.org/draft-04/schema) and object.

YAML Schema

A metaschema extending JSON Schema's metaschema to add support for some YAML-specific constructions.

All of:

`0`

Type: schema (http://json-schema.org/draft-04/schema).

`1`

Type: object.

Properties:

`tag`

Type: string (*len* $\geq$ 6).

A fully-qualified YAML tag name that should be associated with the object type returned by the YAML parser; for example, the object must be an instance of the class registered with the parser to create instances of objects with this tag. Implementation of this validator is optional and depends on details of the YAML parser.

`propertyOrder`

Type: array of (string).

Specifies the default order of the properties when writing out. Any keys not listed in propertyOrder will be in arbitrary order at the end.

Items:

Type: string.

`flow_style`

Type: string from ["block", "flow"].

Specifies the default serialization style to use for an array or object. YAML supports multiple styles for arrays/sequences and objects/maps, called "block style" and "flow style". For example:

```
Block style: !!map
  Clark : Evans
  Ingy  : döt Net
  Oren  : Ben-Kiki

Flow style: !!map { Clark: Evans, Ingy: döt Net, Oren: Ben-Kiki }
```

This property gives a hint to the tool outputting the YAML which style to use. If not provided, the library is free to use whatever heuristics it wishes to determine the output style. This property does not enforce any particular style on YAML being parsed.

`style`

Type: string from ["inline", "literal", "folded"].

Specifies the default serialization style to use for a string. YAML supports multiple styles for strings:

```
Inline style: "First line\nSecond line"

Literal style: |
  First line
  Second line

Folded style: >
  First
  line
```

```
Second
line
```

This property gives a hint to the tool outputting the YAML which style to use. If not provided, the library is free to use whatever heuristics it wishes to determine the output style. This property does not enforce any particular style on YAML being parsed.

`examples`

Type: array of (array).

A list of examples to help document the schema. Each pair is a prose description followed by a string containing YAML content.

Items:

Type: array.

Items:

`index[0]`

Type: string.

`index[1]`

Type: string.

6.1.1 Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
```

```
id: "http://stsci.edu/schemas/yaml-schema/draft-01"
title:
  YAML Schema
description: |
  A metaschema extending JSON Schema's metaschema to add support for
  some YAML-specific constructions.
allOf:
  - $ref: "http://json-schema.org/draft-04/schema"
  - type: object
    properties:
      tag:
        description: |
          A fully-qualified YAML tag name that should be associated
          with the object type returned by the YAML parser; for
          example, the object must be an instance of the class
          registered with the parser to create instances of objects
          with this tag. Implementation of this validator is optional
          and depends on details of the YAML parser.
        type: string
        minLength: 6

      propertyOrder:
        description: |
          Specifies the default order of the properties when writing
          out.  Any keys not listed in propertyOrder will be in
          arbitrary order at the end.
        type: array
        items:
          type: string

      flow_style:
        description: |
          Specifies the default serialization style to use for an
          array or object.  YAML supports multiple styles for
          arrays/sequences and objects/maps, called "block style" and
          "flow style".  For example::

            Block style: !!map
              Clark : Evans
              Ingy  : döt Net
              Oren  : Ben-Kiki

            Flow style: !!map { Clark: Evans, Ingy: döt Net, Oren: Ben-Kiki }

          This property gives a hint to the tool outputting the YAML
          which style to use.  If not provided, the library is free to
          use whatever heuristics it wishes to determine the output
          style.  This property does not enforce any particular style
          on YAML being parsed.
        type: string
        enum: [block, flow]

      style:
        description: |
          Specifies the default serialization style to use for a string.
          YAML supports multiple styles for strings::

            Inline style: "First line\nSecond line"
```

```
62
63            Literal style: |
64              First line
65              Second line
66
67            Folded style: >
68              First
69              line
70
71              Second
72              line
73
74          This property gives a hint to the tool outputting the YAML
75          which style to use.  If not provided, the library is free to
76          use whatever heuristics it wishes to determine the output
77          style.  This property does not enforce any particular style
78          on YAML being parsed.
79        type: string
80        enum: [inline, literal, folded]
81
82      examples:
83        description: |
84          A list of examples to help document the schema.  Each pair
85          is a prose description followed by a string containing YAML
86          content.
87        type: array
88        items:
89          type: array
90          items:
91            - type: string
92            - type: string
93  ...
```

6.2 invoice: Invoice

Type: object.

Invoice

Represents billing invoices.

Definitions:

`address`

Type: object.

An address consisting of a name and street address.

Properties:

`name`

Type: string. Required.

The full name of the addressee (in whatever order is culturally appropriate).

`address`

Type: *definitions/street-address* (page 19). Required.

`street-address`

Type: object.

A street address (excluding the name of the addressee).

Properties:

`lines`

Type: string. Required.

`city`

Type: string. Required.

`state`

Type: string. Required.

`postal`

Type: string (regex `^[0-9]{5}(-[0-9]{4})?$`). Required.

`product`

Type: object.

A listing for a single product on an invoice (including quantity of that product–products with the same SKU should not be listed more than once).

Properties:

`sku`

Type: string. Required.

`quantity`

Type: integer ≥ 1. Required.

`description`

Type: string. Required.

`price`

Type: number ≥ 0. Required.

Properties:

`invoice`

Type: integer ≥ 1.

`date`

Type: string (format date-time).

`bill-to`

Type: *definitions/address* (page 18).

`ship-to`

Type: *definitions/address* (page 18).

`product`

Type: array of (*definitions/product* (page 19)).

Items:

Type: *definitions/product* (page 19).

`tax`

Type: number ≥ 0.

`total`

Type: number ≥ 0.

`comments`

Type: string.

Examples:

An example invoice demonstrating the full schema::

```
%TAG ! tag:stsci.edu:yaml-schema/examples/
--- !invoice
invoice: 34843
date   : 2001-01-23
bill-to: &id001
    - name: Chris Dumars
    - address:
        - lines: |
            458 Walkman Dr.
            Suite #292
        - city: Royal Oak
        - state: MI
        - postal: 48046
ship-to: *id001
product:
    - sku         : BL394D
      quantity    : 4
      description : Basketball
      price       : 450.00
    - sku         : BL4438H
      quantity    : 1
      description : Super Hoop
      price       : 2392.00
tax  : 251.42
total: 4443.52
comments:
    Late afternoon is best.
    Backup contact is Nancy
    Billsmer @ 338-4338.
...
```

6.2.1 Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/yaml-schema/examples/invoice"
tag: "tag:stsci.edu:yaml-schema/examples/invoice"
title: Invoice
description: |
```

```
    Represents billing invoices.

examples:
    -
        - "An example invoice demonstrating the full schema:"
        - |
            %TAG ! tag:stsci.edu:yaml-schema/examples/
            --- !invoice
            invoice: 34843
            date   : 2001-01-23
            bill-to: &id001
                - name: Chris Dumars
                - address:
                    - lines: |
                        458 Walkman Dr.
                        Suite #292
                    - city: Royal Oak
                    - state: MI
                    - postal: 48046
            ship-to: *id001
            product:
                - sku         : BL394D
                  quantity    : 4
                  description : Basketball
                  price       : 450.00
                - sku         : BL4438H
                  quantity    : 1
                  description : Super Hoop
                  price       : 2392.00
            tax  : 251.42
            total: 4443.52
            comments:
                Late afternoon is best.
                Backup contact is Nancy
                Billsmer @ 338-4338.
            ...

type: object

properties:
    invoice:
        type: integer
        minimum: 1
    date:
        type: string
        format: date-time
    bill-to:
        $ref: "#/definitions/address"
    ship-to:
        $ref: "#/definitions/address"
    product:
        type: array
        items:
            $ref: "#/definitions/product"
    tax:
        type: number
        minimum: 0
    total:
```

```
        type: number
        minimum: 0
    comments:
        type: string
        flowStyle: block

definitions:
    address:
        description: |
            An address consisting of a name and street address.
        type: object
        flowStyle: block
        propertyOrder: [name, address]
        required: [name, address]
        properties:
            name:
                type: string
                style: inline
                description:
                    The full name of the addressee (in whatever order is
                    culturally appropriate).
            address:
                $ref: "#/definitions/street-address"
        additionalProperties: false

    street-address:
        description: |
            A street address (excluding the name of the addressee).
        type: object
        flowStyle: block
        propertyOrder: [lines, city, state, postal]
        required: [lines, city, state, postal]
        properties:
            lines:
                type: string
                style: literal
            city:
                type: string
                flowStyle: inline
            state:
                type: string
                flowStyle: inline
            postal:
                type: string
                flowStyle: inline
                pattern: "^[0-9]{5}(-[0-9]{4})?$"
        additionalProperties: false

    product:
        description: |
            A listing for a single product on an invoice (including quantity
            of that product--products with the same SKU should not be listed
            more than once).
        type: object
        flowStyle: block
        required: [sku, quantity, description, price]
        properties:
            sku:
```

```
                type: string
                flowStyle: inline
            quantity:
                type: integer
                minimum: 1
            description:
                type: string
                flowStyle: inline
            price:
                type: number
                minimum: 0
        additionalProperties: false
```