Internet Engineering Task Force	A. Wright, Ed.
Internet-Draft	
Intended status: Informational	H. Andrews, Ed.
Expires: October 23, 2017	Cloudflare, Inc.
	April 21, 2017

JSON Schema: A Media Type for Describing JSON Documents

draft-wright-json-schema-01

Abstract

JSON Schema defines the media type "application/schema+json", a JSON-based format for describing the structure of JSON data. JSON Schema asserts what a JSON document must look like, ways to extract information from it, and how to interact with it, ideal for annotating existing JSON APIs that would not otherwise have hypermedia controls or be machine-readable.

Note to Readers

The issues list for this draft can be found at <https://github.com/json-schema-org/json-schema-spec/issues>.

For additional information, see <http://json-schema.org/>.

To provide feedback, use this issue tracker, the communication methods listed on the homepage, or email the document editors.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 23, 2017.

Copyright Notice

Table of Contents

1. Introduction

JSON Schema is a JSON media type for defining the structure of JSON data. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data.

This specification defines JSON Schema core terminology and mechanisms, including pointing to another JSON Schema by reference, dereferencing a JSON Schema reference, and specifying the vocabulary being used.

Other specifications define the vocabularies that perform assertions about validation, linking, annotation, navigation, and interaction.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The terms "JSON", "JSON text", "JSON value", "member", "element", "object", "array", "number", "string", "boolean", "true", "false", and "null" in this document are to be interpreted as defined in RFC 7159 [RFC7159].

3. Overview

This document proposes a new media type "application/schema+json" to identify a JSON Schema for describing JSON data. JSON Schemas are themselves JSON documents. This, and related specifications, define keywords allowing authors to describe JSON data in several ways.

3.1. Validation

JSON Schema describes the structure of a JSON document (for instance, required properties and length limitations). Applications can use this information to validate instances (check that constraints are met), or inform interfaces to collect user input such that the constraints are satisfied.

Validation behaviour and keywords are specified in a separate document [json-schema-validation].

3.2. Hypermedia and Linking

JSON Hyper-Schema describes the hypertext structure of a JSON document. This includes link relations from the instance to other resources, interpretation of instances as multimedia data, and submission data required to use an API.

Hyper-schema behaviour and keywords are specified in a separate document [json-hyper-schema].

4. Definitions

4.1. JSON Document

A JSON document is an information resource (series of octets) described by the application/json media type.

In JSON Schema, the terms "JSON document", "JSON text", and "JSON value" are interchangeable because of the data model it defines.

JSON Schema is only defined over JSON documents. However, any document or memory structure that can be parsed into or processed according to the JSON Schema data model can be interpreted against a JSON Schema, including media types like CBOR [RFC7049].

4.2. Instance

JSON Schema interprets documents according to a data model. A JSON value interpreted according to this data model is called an "instance".

An instance has one of six primitive types, and a range of possible values depending on the type:

null
: A JSON "null" production

boolean
: A "true" or "false" value, from the JSON "true" or "false" productions

object
: An unordered set of properties mapping a string to an instance, from the JSON "object" production

array
: An ordered list of instances, from the JSON "array" production

number
: An arbitrary-precision, base-10 decimal number value, from the JSON "number" production

string
: A string of Unicode code points, from the JSON "string" production

Whitespace and formatting concerns are thus outside the scope of JSON Schema.

Since an object cannot have two properties with the same key, behavior for a JSON document that tries to define two properties (the "member" production) with the same key (the "string" production) in a single object is undefined.

4.3. Instance equality

Two JSON instances are said to be equal if and only if they are of the same type and have the same value according to the data model. Specifically, this means:

- both are null; or
- both are true; or
- both are false; or
- both are strings, and are the same codepoint-for-codepoint; or
- both are numbers, and have the same mathematical value; or
- both are arrays, and have an equal value item-for-item; or
- both are objects, and each property in one has exactly one property with a key equal to the other's, and that other property has an equal value.

Implied in this definition is that arrays must be the same length, objects must have the same number of members, properties in objects are unordered, there is no way to define multiple properties with the same key, and mere formatting differences (indentation, placement of commas, trailing zeros) are insignificant.

4.4. JSON Schema documents

A JSON Schema document, or simply a schema, is a JSON document used to describe an instance. A schema is itself interpreted as an instance. A JSON Schema MUST be an object or a boolean.

Boolean values are equivalent to the following behaviors:

true
: Always passes validation, as if the empty schema {}

false
: Always fails validation, as if the schema { "not":{} }

Properties that are used to describe the instance are called keywords, or schema keywords. The meaning of properties is specified by the vocabulary that the schema is using.

A JSON Schema MAY contain properties which are not schema keywords. Unknown keywords SHOULD be ignored.

A schema that itself describes a schema is called a meta-schema. Meta-schemas are used to validate JSON Schemas and specify which vocabulary it is using.

An empty schema is a JSON Schema with no properties, or only unknown properties.

4.5. Root schema and subschemas

The root schema is the schema that comprises the entire JSON document in question.

Some keywords take schemas themselves, allowing JSON Schemas to be nested:

```
{
    "title": "root",
    "items": {
        "title": "array item"
```

```
    }
}
```

In this example document, the schema titled "array item" is a subschema, and the schema titled "root" is the root schema.

As with the root schema, a subschema is either an object or a boolean.

5. Fragment identifiers

In accordance with section 3.1 of [RFC6839], the syntax and semantics of fragment identifiers specified for any +json media type SHOULD be as specified for "application/json". (At publication of this document, there is no fragment identification syntax defined for "application/json".)

Additionally, the "application/schema+json" media type supports two fragment identifier structures: plain names and JSON Pointers. The use of JSON Pointers as URI fragment identifiers is described in RFC 6901 [RFC6901]. Fragment identifiers matching the JSON Pointer syntax, including the empty string, MUST be interpreted as JSON Pointer fragment identifiers.

Per the W3C's best practices for fragment identifiers [W3C.WD-fragid-best-practices-20121025], plain name fragment identifiers are reserved for referencing locally named schemas. All fragment identifiers that do not match the JSON Pointer syntax MUST be interpreted as plain name fragment identifiers.

Defining and referencing a plain name fragment identifier in the "$id" keyword [id-keyword] section.

6. General considerations

6.1. Range of JSON values

An instance may be any valid JSON value as defined by JSON [RFC7159]. JSON Schema imposes no restrictions on type: JSON Schema can describe any JSON value, including, for example, null.

6.2. Programming language independence

JSON Schema is programming language agnostic, and supports the full range of values described in the data model. Be aware, however, that some languages and JSON parsers may not be able to represent in memory the full range of values describable by JSON.

6.3. Mathematical integers

Some programming languages and parsers use different internal representations for floating point numbers than they do for integers.

For consistency, integer JSON numbers SHOULD NOT be encoded with a fractional part.

6.4. Extending JSON Schema

Implementations MAY define additional keywords to JSON Schema. Save for explicit agreement, schema authors SHALL NOT expect these additional keywords to be supported by peer implementations. Implementations SHOULD ignore keywords they do not support.

Authors of extensions to JSON Schema are encouraged to write their own meta-schemas, which extend the existing meta-schemas using "allOf". This extended meta-schema SHOULD be referenced using the "$schema" keyword, to allow tools to follow the correct behaviour.

7. The "$schema" keyword

The "$schema" keyword is both used as a JSON Schema version identifier and the location of a resource which is itself a JSON Schema, which describes any schema written for this particular version.

The value of this keyword MUST be a URI [RFC3986] (containing a scheme) and this URI MUST be normalized. The current schema MUST be valid against the meta-schema identified by this URI.

The "$schema" keyword SHOULD be used in a root schema. It MUST NOT appear in subschemas.

[CREF1]

Values for this property are defined in other documents and by other parties. JSON Schema implementations SHOULD implement support for current and previous published drafts of JSON Schema vocabularies as deemed reasonable.

8. Schema references with $ref

The "$ref" keyword is used to reference a schema, and provides the ability to validate recursive structures through self-reference.

An object schema with a "$ref" property MUST be interpreted as a "$ref" reference. The value of the "$ref" property MUST be a URI Reference. Resolved against the current URI base, it identifies the URI of a schema to use. All other properties in a "$ref" object MUST be ignored.

The URI is not a network locator, only an identifier. A schema need not be downloadable from the address if it is a network-addressable URL, and implementations SHOULD NOT assume they should perform a network operation when they encounter a network-addressable URI.

A schema MUST NOT be run into an infinite loop against a schema. For example, if two schemas "#alice" and "#bob" both have an "allOf" property that refers to the other, a naive validator might get stuck in an infinite recursive loop trying to validate the instance. Schemas SHOULD NOT make use of infinite recursive nesting like this; the behavior is undefined.

9. Base URI and dereferencing

9.1. Initial base URI

RFC3986 Section 5.1 [RFC3986] defines how to determine the default base URI of a document.

Informatively, the initial base URI of a schema is the URI at which it was found, or a suitable substitute URI if none is known.

9.2. The "$id" keyword

The "$id" keyword defines a URI for the schema, and the base URI that other URI references within the schema are resolved against. The "$id" keyword itself is resolved against the base URI that the object as a whole appears in.

If present, the value for this keyword MUST be a string, and MUST represent a valid URI-reference [RFC3986]. This value SHOULD be normalized, and SHOULD NOT be an empty fragment <#> or an empty string <>.

The root schema of a JSON Schema document SHOULD contain an "$id" keyword with a URI (containing a scheme). This URI SHOULD either not have a fragment, or have one that is an empty string. [CREF2]

To name subschemas in a JSON Schema document, subschemas can use "$id" to give themselves a document-local identifier. This is done by setting "$id" to a URI reference consisting only of a fragment. The fragment identifier MUST begin with a letter ([A-Za-z]), followed by any number of letters, digits ([0-9]), hyphens ("-"), underscores ("_"), colons (":"), or periods (".").

The effect of defining an "$id" that neither matches the above requirements nor is a valid JSON pointer is not defined.

```
{
    "$id": "http://example.com/root.json",
    "definitions": {
        "A": { "$id": "#foo" },
        "B": {
            "$id": "other.json",
            "definitions": {
                "X": { "$id": "#bar" },
                "Y": { "$id": "t/inner.json" }
            }
        },
        "C": {
            "$id": "urn:uuid:ee564b8a-7a87-4125-8c96-e9f123d6766f"
        }
    }
}
```

For example:

The schemas at the following URI-encoded JSON Pointers [RFC6901] (relative to the root schema) have the following base URIs, and are identifiable by either URI in accordance with Section 5 above:

(document root)
: http://example.com/root.json#

#/definitions/A
: http://example.com/root.json#foo

#/definitions/B
: http://example.com/other.json

#/definitions/B/definitions/X
: http://example.com/other.json#bar

#/definitions/B/definitions/Y
: http://example.com/t/inner.json

#/definitions/C
: urn:uuid:ee564b8a-7a87-4125-8c96-e9f123d6766f

9.2.1. Internal references

Schemas can be identified by any URI that has been given to them, including a JSON Pointer or their URI given directly by "$id".

Tools SHOULD take note of the URIs that schemas, including subschemas, provide for themselves using "$id". This is known as "Internal referencing".

For example, consider this schema:

```
{
    "$id": "http://example.net/root.json",
    "items": {
        "type": "array",
        "items": { "$ref": "#item" }
    },
    "definitions": {
        "single": {
            "$id": "#item",
            "type": "integer"
        }
    }
}
```

When an implementation encounters the <#/definitions/single> schema, it resolves the "$id" URI reference against the current base URI to form <http://example.net/root.json#item>.

When an implementation then looks inside the <#/items> schema, it encounters the <#item> reference, and resolves this to <http://example.net/root.json#item> which is understood as the schema defined elsewhere in the same document.

9.2.2. External references

To differentiate schemas between each other in a vast ecosystem, schemas are identified by URI. As specified above, this does not necessarily mean anything is downloaded, but instead JSON Schema implementations SHOULD already understand the schemas they will be using, including the URIs that identify them.

Implementations SHOULD be able to associate arbitrary URIs with an arbitrary schema and/or automatically associate a schema's "$id"-given URI, depending on the trust that the validator has in the schema.

A schema MAY (and likely will) have multiple URIs, but there is no way for a URI to identify more than one schema. When multiple schemas try to identify with the same URI, validators SHOULD raise an error condition.

10. Usage for hypermedia

JSON has been adopted widely by HTTP servers for automated APIs and robots. This section describes how to enhance processing of JSON documents in a more RESTful manner when used with protocols that support media types and Web linking [RFC5988].

10.1. Linking to a schema

It is RECOMMENDED that instances described by a schema/profile provide a link to a downloadable JSON Schema using the link relation "describedby", as defined by Linked Data Protocol 1.0, section 8.1 [W3C.REC-ldp-20150226].

In HTTP, such links can be attached to any response using the Link header [RFC5988]. An example of such a header would be:

```
Link: <http://example.com/my-hyper-schema#>; rel="describedby"
```

10.2. Describing a profile of JSON

Instances MAY specify a "profile" as described in The 'profile' Link Relation [RFC6906]. When used as a media-type parameter, HTTP servers gain the ability to perform Content-Type Negotiation based on profile. The media-type parameter MUST be a whitespace-separated list of URIs (i.e. relative references are invalid).

The profile URI is opaque and SHOULD NOT automatically be dereferenced. If the implementation does not understand the semantics of the provided profile, the implementation can instead follow the "describedby" links, if any, which may provide information on how to handle the profile. Since "profile" doesn't necessarily point to a network location, the "describedby" relation is used for linking to a downloadable schema. However, for simplicity, schema authors should make these URIs point to the same resource when possible.

In HTTP, the media-type parameter would be sent inside the Content-Type header:

```
Content-Type: application/json;
          profile="http://example.com/my-hyper-schema#"
```

Multiple profiles are whitespace separated:

```
Content-Type: application/json;
          profile="http://example.com/alice http://example.com/bob"
```

HTTP can also send the "profile" in a Link, though this may impact media-type semantics and Content-Type negotiation if this replaces the media-type parameter entirely:

```
Link: </alice>;rel="profile", </bob>;rel="profile"
```

10.3. Usage over HTTP

When used for hypermedia systems over a network, HTTP [RFC7231] is frequently the protocol of choice for distributing schemas. Misbehaving clients can pose problems for server maintainers if they pull a schema over the network more frequently than necessary, when it's instead possible to cache a schema for a long period of time.

HTTP servers SHOULD set long-lived caching headers on JSON Schemas. HTTP clients SHOULD observe caching headers and not re-request documents within their freshness period. Distributed systems SHOULD make use of a shared cache and/or caching proxy.

```
User-Agent: product-name/5.4.1 so-cool-json-schema/1.0.2 curl/7.43.0
```

Clients SHOULD set or prepend a User-Agent header specific to the JSON Schema implementation or software product. Since symbols are listed in decreasing order of significance, the JSON Schema library name/version should precede the more generic HTTP library name (if any). For example:

Clients SHOULD be able to make requests with a "From" header so that server operators can contact the owner of a potentially misbehaving script.

11. Security considerations

Both schemas and instances are JSON values. As such, all security considerations defined in RFC 7159 [RFC7159] apply.

Instances and schemas are both frequently written by untrusted third parties, to be deployed on public Internet servers. Validators should take care that the parsing of schemas doesn't consume excessive system resources. Validators MUST NOT fall into an infinite loop.

Servers need to take care that malicious parties can't change the functionality of existing schemas by uploading a schema with an pre-existing or very similar "$id".

Individual JSON Schema vocabularies are liable to also have their own security considerations. Consult the respective specifications for more information.

12. IANA Considerations

The proposed MIME media type for JSON Schema is defined as follows:

Type name: application

Subtype name: schema+json

Required parameters: N/A

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See JSON [RFC7159].

Security considerations: See Section 11 above.

Interoperability considerations: See Sections 6.2 and 6.3 above.

Fragment identifier considerations: See Section 5

13. References

13.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
[RFC3986]	Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005.
[RFC6839]	Hansen, T. and A. Melnikov, "Additional Media Type Structured Syntax Suffixes", RFC 6839, DOI 10.17487/RFC6839, January 2013.
[RFC6901]	Bryan, P., Zyp, K. and M. Nottingham, "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013.
[RFC7159]	Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.
[W3C.REC-ldp-20150226]	Speicher, S., Arwe, J. and A. Malhotra, "Linked Data Platform 1.0", World Wide Web Consortium Recommendation REC-ldp-20150226, February 2015.

13.2. Informative References

[RFC5988]	Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010.
[RFC6906]	Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013.
[RFC7049]	Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013.
[RFC7231]	Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014.
[W3C.WD-fragid-best-practices-20121025]	Tennison, J., "Best Practices for Fragment Identifiers and Media Type Definitions", World Wide Web Consortium LastCall WD-fragid-best-practices-20121025, October 2012.
[json-schema-validation]	Wright, A. and G. Luff, "JSON Schema Validation: A Vocabulary for Structural Validation of JSON", Internet-Draft draft-wright-json-schema-validation-00, October 2016.
[json-hyper-schema]	Wright, A. and G. Luff, "JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON", Internet-Draft draft-wright-json-schema-hyperschema-00, October 2016.

Appendix A. Acknowledgments

Thanks to Gary Court, Francis Galiegue, Kris Zyp, and Geraint Luff for their work on the initial drafts of JSON Schema.

Thanks to Jason Desrosiers, Daniel Perrett, Erik Wilde, Ben Hutton, Evgeny Poberezkin, Brad Bowman, Gowry Sankar, Donald Pipowitch, and Dave Finlay for their submissions and patches to the document.

Appendix B. ChangeLog

[CREF3]

draft-wright-json-schema-01

- Updated intro
- Allowed for any schema to be a boolean
- "$schema" SHOULD NOT appear in subschemas, although that may change
- Changed "id" to "$id"; all core keywords prefixed with "$"
- Clarify and formalize fragments for application/schema+json
- Note applicability to formats such as CBOR that can be represented in the JSON data model

draft-wright-json-schema-00

- Updated references to JSON
- Updated references to HTTP
- Updated references to JSON Pointer
- Behavior for "id" is now specified in terms of RFC3986
- Aligned vocabulary usage for URIs with RFC3986
- Removed reference to draft-pbryan-zyp-json-ref-03
- Limited use of "$ref" to wherever a schema is expected
- Added definition of the "JSON Schema data model"
- Added additional security considerations
- Defined use of subschema identifiers for "id"
- Rewrote section on usage with HTTP
- Rewrote section on usage with rel="describedBy" and rel="profile"
- Fixed numerous invalid examples

draft-zyp-json-schema-04

- Split validation keywords into separate document

draft-zyp-json-schema-00

- Initial draft.
- Salvaged from draft v3.
- Mandate the use of JSON Reference, JSON Pointer.
- Define the role of "id". Define URI resolution scope.
- Add interoperability considerations.

Authors' Addresses

Austin Wright (editor)
EMail: aaa@bzfx.net

Henry Andrews (editor)
Cloudflare, Inc.
EMail: henry@cloudflare.com