Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 23, 2018
November 19, 2017

A. Wright, Ed.

H. Andrews, Ed.
Cloudflare, Inc.

JSON Schema: A Media Type for Describing JSON Documents
draft-handrews-json-schema-00

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. The "application/schema-instance+json" media type provides additional feature-rich integration with "application/schema+json" beyond what can be offered for "application/json" documents.

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

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. It also proposes a further optional media type, "application/schema-instance+json", to provide additional integration features. 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

A JSON document to which a schema is applied is known as an "instance".

4.2.1. Instance Data Model

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, including different lexical representations of numbers that are equal within the data model, are thus outside the scope of JSON Schema. JSON Schema vocabularies (Section 4.3.2) that wish to work with such differences in lexical representations SHOULD define keywords to precisely interpret formatted strings within the data model rather than relying on having the original JSON representation Unicode characters available.

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.

Note that JSON Schema vocabularies are free to define their own extended type system. This should not be confused with the core data model types defined here. As an example, "integer" is a reasonable type for a vocabulary to define as a value for a keyword, but the data model makes no distinction between integers and other numbers.

4.2.2. Instance Media Types

JSON Schema is designed to fully work with "application/json" documents, as well as media types using the "+json" structured syntax suffix.

Some functionality that is useful for working with schemas is defined by each media type, namely media type parameters and URI fragment identifier syntax and semantics. These features are useful in

content negotiation and in calculating URIs for specific locations within an instance, respectively.

This specification defines the "application/schema-instance+json" media type in order to allow instance authors to take full advantage of parameters and fragment identifiers for these purposes.

4.2.3. 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.

4.3. 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, but SHOULD always be given the media type "application/schema+json" rather than "application/schema-instance+json". The "application/schema+json" media type is defined to offer a superset of the media type parameter and fragment identifier syntax and semantics provided by "application/schema-instance+json".

4.3.1. JSON Schema Values and Keywords

A JSON Schema MUST be an object or a boolean.

Object properties that are applied to the instance are called keywords, or schema keywords. Broadly speaking, keywords fall into one or both of two categories:

assertions produce a boolean result when applied to an instance

annotations attach information to an instance for application use

Keywords may fall into either or both categories. Extension keywords, meaning those defined outside of this document and its companions, are free to define other behaviors as well.

The boolean schema values "true" and "false" are trivial assertions that always return themselves regardless of the instance value. As an example, in terms of the validation vocabulary, boolean schemas are equivalent to the following behaviors:

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

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

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

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

4.3.2. JSON Schema Vocabularies

A JSON Schema vocabulary is a set of keywords defined for a particular purpose. The vocabulary specifies the meaning of its keywords as assertions, annotations, and/or any vocabulary-defined keyword category. The two companion standards to this document each define a vocabulary: One for instance validation, and one for hypermedia annotations. Vocabularies are the primary mechanism for extensibility within the JSON Schema media type.

Vocabularies may be defined by any entity. Vocabulary authors SHOULD take care to avoid keyword name collisions if the vocabulary is intended for broad use, and potentially combined with other vocabularies. JSON Schema does not provide any formal namespacing system, but also does not constrain keyword names, allowing for any number of namespacing approaches.

Vocabularies may build on each other, such as by defining the behavior of their keywords with respect to the behavior of keywords from another vocabulary, or by using a keyword from another vocabulary with a restricted or expanded set of acceptable values. Not all such vocabulary re-use will result in a new vocabulary that is compatible with the vocabulary on which it is built. Vocabulary authors SHOULD clearly document what level of compatibility, if any, is expected.

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. [[CREF1: Currently, only a single meta-schema may be specified per schema, meaning that in order to use multiple vocabularies, a meta-schema must be written that encompasses all of them. The hyper-schema meta-schema is an example of this, as it encompasses the validation vocabulary as well as the hypermedia vocabulary.]]

4.3.3. 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 "application/schema-instance+json" media type supports one fragment identifier structure: JSON Pointers.

The use of JSON Pointers as URI fragment identifiers is described in RFC 6901 [RFC6901]. For "application/schema+json", which supports two fragment identifier syntaxes, 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 in "application/schema+json" 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 within an "application/schema+json" document are specified in the "$id" keyword (Section 9.2) 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.

Note that the recursive nature of meta-schemas requires re-defining recursive keywords in the extended meta-schema, as can be seen in the JSON Hyper-Schema meta-schema.

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.

If this URI identifies a retrievable resource, that resource SHOULD be of media type "application/schema+json".

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

[[CREF2: Using multiple "$schema" keywords in the same document would imply that the vocabulary and therefore behavior can change within a document. This would necessitate resolving a number of implementation concerns that have not yet been clearly defined. So, while the pattern of using "$schema" only in root schemas is likely to remain the best practice for schema authoring, implementation behavior is subject to be revised or liberalized in future drafts.]]

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 "$id" keyword defines a URI for the schema, and the base URI that other URI references within the schema are resolved against. A subschema's "$id" is resolved against the base URI of its parent schema. If no parent sets an explicit base with "$id", the base URI is that of the entire document, as determined per RFC 3986 section 5 [RFC3986].

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 "$id" keyword with a URI (containing a scheme). This URI SHOULD either not have a fragment, or have one that is an empty string. [[CREF3: How should an "$id" URI reference containing a fragment with other components be interpreted? There are two cases: when the other components match the current base URI and when they change the base URI.]]

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 plain name fragment (not a JSON Pointer 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 (".").

Providing a plain name fragment enables a subschema to be relocated within a schema without requiring that JSON Pointer references are updated.

The effect of defining a fragment-only "$id" URI reference that neither matches the above requirements nor is a valid JSON pointer is not defined.

For example:

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

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 without needing to resolve the fragment against the base URI.

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. Comments With "$comment"

This keyword is reserved for comments from schema authors to readers or maintainers of the schema. The value of this keyword MUST be a string. Implementations MUST NOT present this string to end users. Tools for editing schemas SHOULD support displaying and editing this keyword. The value of this keyword MAY be used in debug or error output which is intended for developers making use of schemas. Schema vocabularies SHOULD allow "$comment" within any object containing vocabulary keywords. Implementations MAY assume "$comment" is allowed unless the vocabulary specifically forbids it. Vocabularies MUST NOT specify any effect of "$comment" beyond what is described in this specification. Tools that translate other media types or programming languages to and from application/schema+json MAY choose to convert that media type or programming language's native comments to or from "$comment" values. The behavior of such translation when both native comments and "$comment" properties are present is implementation-dependent. Implementations SHOULD treat "$comment" identically to an unknown extension keyword. They MAY strip "$comment" values at any point during processing. In particular, this allows for shortening schemas when the size of deployed schemas is a concern. Implementations MUST NOT take any other action based on the presence, absence, or contents of "$comment" properties.

11. 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 [RFC8288].

11.1. Linking to a Schema

It is RECOMMENDED that instances described by a schema 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 [RFC8288]. An example of such a header would be:

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

11.2. Identifying a Schema via a Media Type Parameter

Media types MAY allow for a "schema" media type parameter, which gives HTTP servers the ability to perform Content-Type Negotiation based on schema. The media-type parameter MUST be a whitespace-separated list of URIs (i.e. relative references are invalid).

When using the media type application/schema-instance+json, the "schema" parameter MUST be supplied.

The schema URI is opaque and SHOULD NOT automatically be dereferenced. If the implementation does not understand the semantics of the provided schema, the implementation can instead follow the "describedby" links, if any, which may provide information on how to handle the schema. Since "schema" 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;
          schema="http://example.com/my-hyper-schema#"
```

Multiple schemas are whitespace separated:

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

[[CREF4: This paragraph assumes that we can register a "schema" link relation. Should we instead specify something like "tag:json-schema.org,2017:schema" for now?]] HTTP can also send the "schema" 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="schema", </bob>;rel="schema"
```

11.3. 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.

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:

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

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

12. 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.

Schema authors should take care with "$comment" contents, as a malicious implementation can display them to end-users in violation of a spec, or fail to strip them if such behavior is expected.

A malicious schema author could place executable code or other dangerous material within a "$comment". Implementations MUST NOT parse or otherwise take action based on "$comment" contents.

13. IANA Considerations

13.1. application/schema+json

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 12 above.

Interoperability considerations: See Sections 6.2 and 6.3 above.

Fragment identifier considerations: See Section 5

13.2. application/schema-instance+json

The proposed MIME media type for JSON Schema Instances that require a JSON Schema-specific media type is defined as follows:

Type name: application

Subtype name: schema-instance+json

Required parameters:

schema A non-empty list of space-separated URIs, each identifying a JSON Schema resource. The instance SHOULD successfully validate against at least one of these schemas. Non-validating schemas MAY be included for purposes such as allowing clients to make use of older versions of a schema as long as the runtime instance validates against that older version.

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

Security considerations: See Section 12 above.

Interoperability considerations: See Sections 6.2 and 6.3 above.

Fragment identifier considerations: See Section 5

14. References

14.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[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, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6839] Hansen, T. and A. Melnikov, "Additional Media Type Structured Syntax Suffixes", RFC 6839, DOI 10.17487/RFC6839, January 2013, <https://www.rfc-editor.org/info/rfc6839>.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/info/rfc6901>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[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, <http://www.w3.org/TR/2015/REC-ldp-20150226>.

14.2. Informative References

[json-hyper-schema]
Andrews, H. and A. Wright, "JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON", draft-handrews-json-schema-hyperschema-00 (work in progress), November 2017.

[json-schema-validation]
Wright, A., Andrews, H., and G. Luff, "JSON Schema Validation: A Vocabulary for Structural Validation of JSON", draft-handrews-json-schema-validation-00 (work in progress), November 2017.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[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, <http://www.w3.org/TR/2012/WD-fragid-best-practices-20121025>.

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

[[CREF5: This section to be removed before leaving Internet-Draft status.]]

draft-handrews-json-schema-00

- Make the concept of a schema keyword vocabulary more clear
- Note that the concept of "integer" is from a vocabulary, not the data model
- Classify keywords as assertions or annotations and describe their general behavior
- Explain the boolean schemas in terms of generalized assertions
- Reserve "$comment" for non-user-visible notes about the schema
- Wording improvements around "$id" and fragments
- Note the challenges of extending meta-schemas with recursive references
- Add "application/schema-instance+json" media type
- Recommend a "schema" link relation / parameter instead of "profile"

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.
San Francisco, CA
USA

EMail: henry@cloudflare.com