Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: August 4, 2013

F. Galiegue, Ed.

K. Zyp, Ed.
SitePen (USA)

G. Court

January 31, 2013

JSON Schema: core definitions and terminology
draft-zyp-json-schema-04

Abstract

JSON Schema defines the media type "application/schema+json", a JSON based format for defining the structure of JSON data. JSON Schema provides a contract for what JSON data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data.

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 August 4, 2013.

Copyright Notice

Table of Contents

1. Introduction

JSON Schema is a JSON media type for defining the structure of JSON data. JSON Schema provides a contract for what JSON data is required for a given application and how to interact with it. 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; related specifications build upon this specification and define different applications of JSON Schema.

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 4627 [RFC4627].

3. Core terminology

3.1. Property, item

When refering to a JSON Object, as defined by [RFC4627], the terms "member" and "property" may be used interchangeably.

When refering to a JSON Array, as defined by [RFC4627], the terms "element" and "item" may be used interchangeably.

3.2. JSON Schema, keywords

A JSON Schema is a JSON document, and that document MUST be an object. Object members (or properties) defined by JSON Schema (this specification, or related specifications) are called keywords, or schema keywords.

A JSON Schema MAY contain properties which are not schema keywords.

3.3. Empty schema

An empty schema is a JSON Schema with no properties, or with properties which are not schema keywords.

3.4. Root schema, subschema

This example of a JSON Schema has no subschemas:

```
{
    "title": "root"
}
```

JSON Schemas can also be nested, as in this example:

```
{
    "title": "root",
    "otherSchema": {
        "title": "nested",
        "anotherSchema": {
            "title": "alsoNested"
        }
    }
}
```

In this example, "nested" and "alsoNested" are subschemas, and "root" is a root schema.

3.5. JSON Schema primitive types

JSON Schema defines seven primitive types for JSON values:

array A JSON array.

boolean A JSON boolean.

integer A JSON number without a fraction or exponent part.

number Any JSON number. Number includes integer.

null The JSON null value.

object A JSON object.

string A JSON string.

3.6. JSON value equality

Two JSON values are said to be equal if and only if:

- both are nulls; or
- both are booleans, and have the same value; or
- both are strings, and have the same value; or
- both are numbers, and have the same mathematical value; or
- both are arrays, and:
  - have the same number of items; and
  - items at the same index are equal according to this definition; or
- both are objects, and:
  - have the same set of property names; and
  - values for a same property name are equal according to this definition.

3.7. Instance

An instance is any JSON value. An instance may be described by one or more schemas.

An instance may also be referred to as "JSON instance", or "JSON data".

4. Overview

This document proposes a new media type "application/schema+json" to identify JSON Schema for describing JSON data. JSON Schemas are themselves written in JSON. This, and related specifications, define keywords allowing to describe this data in terms of allowable values, textual descriptions and interpreting relations with other resources. The following sections are a summary of features defined by related specifications.

4.1. Validation

JSON Schema allows applications to validate instances, either non interactively or interactively. For instance, an application may

collect JSON data and check that this data matches a given set of constraints; another application may use a JSON Schema to build an interactive interface in order to collect user input according to constraints described by JSON Schema.

4.2. Hypermedia and linking

JSON Schema provides a method for extracting link relations from instances to other resources, as well as describing interpretations of instances as multimedia data. This allows JSON data to be interpreted as rich hypermedia documents, placed in the context of a larger set of related resources.

5. General considerations

5.1. Applicability to all JSON values

It is acknowledged that an instance may be any valid JSON value as defined by [RFC4627]. As such, JSON Schema does not mandate that an instance be of a particular type: JSON Schema can describe any JSON value, including null.

5.2. Programming language independence

JSON Schema is programming language agnostic. The only limitations are the ones expressed by [RFC4627] and those of the host programming language.

5.3. JSON Schema and HTTP

This specification acknowledges the role of HTTP [RFC2616] as the dominant protocol in use on the Internet, and the wealth of official specifications related to it.

This specification uses a subset of these specifications to recommend a set of mechanisms, usable by this protocol, to associate JSON instances to one or more schemas.

5.4. JSON Schema and other protocols

JSON Schema does not define any semantics for the client-server interface for any other protocols than HTTP. These semantics are application dependent, or subject to agreement between the parties involved in the use of JSON Schema for their own needs.

5.5. Mathematical integers

It is acknowledged by this specification that some programming languages, and their associated parsers, use different internal representations for floating point numbers and integers, while others do not.

As a consequence, for interoperability reasons, JSON values used in the context of JSON Schema, whether that JSON be a JSON Schema or an instance, SHOULD ensure that mathematical integers be represented as integers as defined by this specification.

5.6. Extending JSON Schema

Implementations MAY choose to 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.

5.7. Security considerations

Both schemas and instances are JSON values. As such, all security considerations defined in RFC 4627 [RFC4627] apply.

6. The "$schema" keyword

6.1. Purpose

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.

This keyword MUST be located at the root of a JSON Schema. The value of this keyword MUST be a URI [RFC3986] and a valid JSON Reference [json-reference]; this URI MUST be both absolute and normalized. The resource located at this URI MUST successfully describe itself. It is RECOMMENDED that schema authors include this keyword in their schemas.

The following values are predefined:

http://json-schema.org/schema# JSON Schema written against the current version of the specification.

http://json-schema.org/hyper-schema# JSON Schema written against the current version of the specification.

http://json-schema.org/draft-04/schema# JSON Schema written against this version.

http://json-schema.org/draft-04/hyper-schema# JSON Schema hyperschema written against this version.

http://json-schema.org/draft-03/schema# JSON Schema written against JSON Schema, draft v3 [json-schema-03].

http://json-schema.org/draft-03/hyper-schema# JSON Schema hyperschema written against JSON Schema, draft v3 [json-schema-03].

6.2. Customization

When extending JSON Schema with custom keywords, schema authors SHOULD define a custom URI for "$schema". This custom URI MUST NOT be one of the predefined values.

7. URI resolution scopes and dereferencing

7.1. Definition

JSON Schema uses JSON Reference [json-reference] as a mechanism for schema addressing. It extends this specification in two ways:

JSON Schema offers facilities to alter the base URI against which a reference must resolve by the means of the "id" keyword;

it defines a specific dereferencing mechanism extending JSON Reference to accept arbitrary fragment parts.

Altering the URI within a schema is called defining a new resolution scope. The initial resolution scope of a schema is the URI of the schema itself, if any, or the empty URI if the schema was not loaded from a URI.

7.2. URI resolution scope alteration with the "id" keyword

7.2.1. Valid values

The value for this keyword MUST be a string, and MUST be a valid URI. This URI MUST be normalized, and SHOULD NOT be an empty fragment (#) or the empty URI.

7.2.2. Usage

The "id" keyword (or "id", for short) is used to alter the resolution scope. When an id is encountered, an implementation MUST resolve this id against the most immediate parent scope. The resolved URI will be the new resolution scope for this subschema and all its children, until another id is encountered.

When using "id" to alter resolution scopes, schema authors SHOULD ensure that resolution scopes are unique within the schema.

This schema will be taken as an example:

```
{
    "id": "http://x.y.z/rootschema.json#",
    "schema1": {
        "id": "#foo"
    },
    "schema2": {
        "id": "otherschema.json",
        "nested": {
            "id": "#bar"
        },
        "alsonested": {
            "id": "t/inner.json#a"
        }
    },
    "schema3": {
        "id": "some://where.else/completely#"
    }
}
```

Subschemas at the following URI-encoded JSON Pointer [json-pointer]s (starting from the root schema) define the following resolution scopes:

(document root) http://x.y.z/rootschema.json#

#/schema1 http://x.y.z/rootschema.json#foo

#/schema2 http://x.y.z/otherschema.json#

#/schema2/nested http://x.y.z/otherschema.json#bar

#/schema2/alsonested http://x.y.z/t/inner.json#a

#/schema3 some://where.else/completely#

7.2.3. Canonical dereferencing and inline dereferencing

When resolving a URI against a resolution scope, an implementation may choose two modes of operation:

canonical dereferencing The implementation dereferences all resolved URIs.

inline dereferencing The implementation chooses to dereference URIs within the schema.

Implementations MUST support canonical dereferencing, and MAY support inline dereferencing.

For example, consider this schema:

```
{
    "id": "http://my.site/myschema#",
    "definitions": {
        "schema1": {
            "id": "schema1",
            "type": "integer"
        },
        "schema2", {
            "type": "array",
            "items": { "$ref": "schema1" }
        }
    }
}
```

When an implementation encounters the "schema1" reference, it resolves it against the most immediate parent scope, leading to URI "http://my.site/schema1#". The way to process this URI will differ according to the chosen dereferencing mode:

if canonical dereferencing is used, the implementation will dereference this URI, and fetch the content at this URI;

if inline dereferencing is used, the implementation will notice that URI scope "http://my.site/schema1#" is already defined within the schema, and choose to use the appropriate subschema.

7.2.4. Inline dereferencing and fragments

When using inline dereferencing, a resolution scope may lead to a URI which has a non empty fragment part which is not a JSON Pointer, as in this example:

```
{
    "id": "http://some.site/schema#",
    "not": { "$ref": "#inner" },
    "definitions": {
        "schema1": {
            "id": "#inner",
            "type": "boolean"
        }
    }
}
```

An implementation choosing to support inline dereferencing SHOULD be able to use this kind of reference. Implementations choosing to use canonical dereferencing, however, are not required to support it.

7.3. Interoperability considerations

Inline dereferencing can produce canonical URIs which differ from the canonical URI of the root schema. Schema authors SHOULD ensure that implementations using canonical dereferencing obtain the same content as implementations using inline dereferencing.

Extended JSON References using fragments which are not JSON Pointers are not dereferenceable by implementations choosing not to support inline dereferencing. This kind of reference is defined for backwards compatibility, and SHOULD NOT be used in new schemas.

8. Recommended correlation mechanisms for use with the HTTP protocol

It is acknowledged by this specification that the majority of interactive JSON Schema processing will be over HTTP. This section therefore gives recommendations for materializing an instance/schema correlation using mechanisms currently available for this protocol. An instance is said to be described by one (or more) schema(s).

8.1. Correlation by means of the "Content-Type" header

It is RECOMMENDED that a MIME type parameter by the name of "profile" be appended to the "Content-Type" header of the instance being processed. If present, the value of this parameter MUST be a valid

URI, and this URI SHOULD resolve to a valid JSON Schema. The MIME type MUST be "application/json", or any other subtype.

An example of such a header would be:

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

8.2. Correlation by means of the "Link" header

When using the "Link" header, the relation type used MUST be "describedBy", as defined by RFC 5988, section 5.3 [RFC5988]. The target URI of the "Link" header MUST be a valid JSON Schema.

An example of such a header would be:

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

9. IANA Considerations

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

type name: application;

subtype name: schema+json.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[json-reference] Bryan, P. and K. Zyp, "JSON Reference (work in progress)", September 2012, <http://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03>.

[json-pointer] Bryan, P. and K. Zyp, "JSON Pointer (work in progress)", September 2012, <http://tools.ietf.org/html/draft-ietf-appsawg-json-pointer-07>.

[json-schema-03] Court, G. and K. Zyp, "JSON Schema, draft 3", September 2012, <http://tools.ietf.org/html/draft-zyp-json-schema-03>.

Appendix A. ChangeLog

draft-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

Francis Galiegue (editor)

EMail: fgaliegue@gmail.com

Kris Zyp (editor)
SitePen (USA)
530 Lytton Avenue
Palo Alto, CA 94301
USA

Phone: +1 650 968 8787
EMail: kris@sitepen.com

Gary Court
Calgary, AB
Canada

EMail: gary.court@gmail.com