Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 16, 2017

A. Wright, Ed.

G. Luff

October 13, 2016

JSON Schema Validation: A Vocabulary for Structural Validation of JSON
draft-wright-json-schema-validation-00

Abstract

JSON Schema (application/schema+json) has several purposes, one of which is JSON instance validation. This document specifies a vocabulary for JSON Schema to describe the meaning of JSON documents, provide hints for user interfaces working with JSON data, and to make assertions about what a valid document must look like.

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 April 16, 2017.

Table of Contents

1. Introduction

JSON Schema can be used to require that a given JSON document (an instance) satisfies a certain number of criteria. These criteria are asserted by using keywords described in this specification. In addition, a set of keywords is also defined to assist in interactive, user interface instance generation.

This specification will use the terminology defined by the JSON Schema core [json-schema] specification.

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].

This specification uses the term "container instance" to refer to both array and object instances. It uses the term "children instances" to refer to array elements or object member values.

This specification uses the term "property set" to refer to the set of an object's member names; for instance, the property set of JSON Object { "a": 1, "b": 2 } is ["a", "b"].

Elements in an array value are said to be unique if no two elements of this array are equal [json-schema].

3. Interoperability considerations

3.1. Validation of string instances

It should be noted that the nul character (\u0000) is valid in a JSON string. An instance to validate may contain a string value with this character, regardless of the ability of the underlying programming language to deal with such data.

3.2. Validation of numeric instances

The JSON specification allows numbers with arbitrary precision, and JSON Schema does not add any such bounds. This means that numeric instances processed by JSON Schema can be arbitrarily large and/or have an arbitrarily long decimal part, regardless of the ability of the underlying programming language to deal with such data.

3.3. Regular expressions

Two validation keywords, "pattern" and "patternProperties", use regular expressions to express constraints. These regular expressions SHOULD be valid according to the ECMA 262 [ecma262] regular expression dialect.

Furthermore, given the high disparity in regular expression constructs support, schema authors SHOULD limit themselves to the following regular expression tokens:

- individual Unicode characters, as defined by the JSON specification [RFC7159];
- simple character classes ([abc]), range character classes ([a-z]);
- complemented character classes ([^abc], [^a-z]);
- simple quantifiers: "+" (one or more), "*" (zero or more), "?" (zero or one), and their lazy versions ("+?", "*?", "??");

range quantifiers: "{x}" (exactly x occurrences), "{x,y}" (at least x, at most y, occurrences), {x,} (x occurrences or more), and their lazy versions;

the beginning-of-input ("^") and end-of-input ("$") anchors;

simple grouping ("(...)") and alternation ("|").

Finally, implementations MUST NOT take regular expressions to be anchored, neither at the beginning nor at the end. This means, for instance, the pattern "es" matches "expression".

4. General validation considerations

4.1. Keywords and instance primitive types

Most validation keywords only limit the range of values within a certain primitive type. When the primitive type of the instance is not of the type targeted by the keyword, the validation succeeds.

For example, the "maxLength" keyword will only restrict certain strings (that are too long) from being valid. If the instance is a number, boolean, null, array, or object, the keyword passes validation.

4.2. Missing keywords

Validation keywords that are missing never restrict validation. In some cases, this no-op behavior is identical to a keyword that exists with certain values, and these values are noted where known.

4.3. Linearity

Validation keywords typically operate independent of each other, without affecting each other.

For author convienence, there are some exceptions:

"additionalProperties", whose behavior is defined in terms of "properties" and "patternProperties"; and

"additionalItems", whose behavior is defined in terms of "items"

5. Validation keywords

Validation keywords in a schema impose requirements for successfully validating an instance.

5.1. multipleOf

The value of "multipleOf" MUST be a number, strictly greater than 0.

A numeric instance is only valid if division by this keyword's value results in an integer.

5.2. maximum

The value of "maximum" MUST be a number, representing an upper limit for a numeric instance.

If the instance is a number, then this keyword validates if "exclusiveMaximum" is true and instance is less than the provided value, or else if the instance is less than or exactly equal to the provided value.

5.3. exclusiveMaximum

The value of "exclusiveMaximum" MUST be a boolean, representing whether the limit in "maximum" is exclusive or not. An undefined value is the same as false.

If "exclusiveMaximum" is true, then a numeric instance SHOULD NOT be equal to the value specified in "maximum". If "exclusiveMaximum" is false (or not specified), then a numeric instance MAY be equal to the value of "maximum".

5.4. minimum

The value of "minimum" MUST be a number, representing a lower limit for a numeric instance.

If the instance is a number, then this keyword validates if "exclusiveMinimum" is true and instance is greater than the provided value, or else if the instance is greater than or exactly equal to the provided value.

5.5. exclusiveMinimum

The value of "exclusiveMinimum" MUST be a boolean, representing whether the limit in "minimum" is exclusive or not. An undefined value is the same as false.

If "exclusiveMinimum" is true, then a numeric instance SHOULD NOT be equal to the value specified in "minimum". If "exclusiveMinimum" is false (or not specified), then a numeric instance MAY be equal to the value of "minimum".

5.6. maxLength

The value of this keyword MUST be a non-negative integer.

The value of this keyword MUST be an integer. This integer MUST be greater than, or equal to, 0.

A string instance is valid against this keyword if its length is less than, or equal to, the value of this keyword.

The length of a string instance is defined as the number of its characters as defined by RFC 7159 [RFC7159].

5.7. minLength

A string instance is valid against this keyword if its length is greater than, or equal to, the value of this keyword.

The length of a string instance is defined as the number of its characters as defined by RFC 7159 [RFC7159].

The value of this keyword MUST be an integer. This integer MUST be greater than, or equal to, 0.

"minLength", if absent, may be considered as being present with integer value 0.

5.8. pattern

The value of this keyword MUST be a string. This string SHOULD be a valid regular expression, according to the ECMA 262 regular expression dialect.

A string instance is considered valid if the regular expression matches the instance successfully. Recall: regular expressions are not implicitly anchored.

5.9. additionalItems and items

The value of "additionalItems" MUST be either a boolean or an object. If it is an object, this object MUST be a valid JSON Schema.

The value of "items" MUST be either a schema or array of schemas.

Successful validation of an array instance with regards to these two keywords is determined as follows:

if "items" is not present, or its value is an object, validation of the instance always succeeds, regardless of the value of "additionalItems";

if the value of "additionalItems" is boolean value true or an object, validation of the instance always succeeds;

if the value of "additionalItems" is boolean value false and the value of "items" is an array, the instance is valid if its size is less than, or equal to, the size of "items".

If either keyword is absent, it may be considered present with an empty schema.

5.10. maxItems

The value of this keyword MUST be an integer. This integer MUST be greater than, or equal to, 0.

An array instance is valid against "maxItems" if its size is less than, or equal to, the value of this keyword.

5.11. minItems

The value of this keyword MUST be an integer. This integer MUST be greater than, or equal to, 0.

An array instance is valid against "minItems" if its size is greater than, or equal to, the value of this keyword.

If this keyword is not present, it may be considered present with a value of 0.

5.12. uniqueItems

The value of this keyword MUST be a boolean.

If this keyword has boolean value false, the instance validates successfully. If it has boolean value true, the instance validates successfully if all of its elements are unique.

If not present, this keyword may be considered present with boolean value false.

5.13. maxProperties

The value of this keyword MUST be an integer. This integer MUST be greater than, or equal to, 0.

An object instance is valid against "maxProperties" if its number of properties is less than, or equal to, the value of this keyword.

5.14. minProperties

The value of this keyword MUST be an integer. This integer MUST be greater than, or equal to, 0.

An object instance is valid against "minProperties" if its number of properties is greater than, or equal to, the value of this keyword.

If this keyword is not present, it may be considered present with a value of 0.

5.15. required

The value of this keyword MUST be an array. This array MUST have at least one element. Elements of this array MUST be strings, and MUST be unique.

An object instance is valid against this keyword if its property set contains all elements in this keyword's array value.

5.16. properties

The value of "properties" MUST be an object. Each value of this object MUST be an object, and each object MUST be a valid JSON Schema.

If absent, it can be considered the same as an empty object.

5.17. patternProperties

The value of "patternProperties" MUST be an object. Each property name of this object SHOULD be a valid regular expression, according to the ECMA 262 regular expression dialect. Each property value of this object MUST be an object, and each object MUST be a valid JSON Schema.

If absent, it can be considered the same as an empty object.

5.18. additionalProperties

The value of "additionalProperties" MUST be a boolean or a schema.

If "additionalProperties" is absent, it may be considered present with an empty schema as a value.

If "additionalProperties" is true, validation always succeeds.

If "additionalProperties" is false, validation succeeds only if the instance is an object and all properties on the instance were covered by "properties" and/or "patternProperties".

If "additionalProperties" is an object, validate the value as a schema to all of the properties that weren't validated by "properties" nor "patternProperties".

5.19. dependencies

This keyword specifies rules that are evaluated if the instance is an object and contains a certain property.

This keyword's value MUST be an object. Each property specifies a dependency. Each dependency value MUST be an object or an array.

If the dependency value is an object, it MUST be a valid JSON Schema. If the dependency key is a property in the instance, the dependency value must validate against the entire instance.

If the dependency value is an array, it MUST have at least one element, each element MUST be a string, and elements in the array MUST be unique. If the dependency key is a property in the instance, each of the items in the dependency value must be a property that exists in the instance.

5.20. enum

The value of this keyword MUST be an array. This array SHOULD have at least one element. Elements in the array SHOULD be unique.

Elements in the array MAY be of any type, including null.

An instance validates successfully against this keyword if its value is equal to one of the elements in this keyword's array value.

5.21. type

The value of this keyword MUST be either a string or an array. If it is an array, elements of the array MUST be strings and MUST be unique.

String values MUST be one of the seven primitive types defined by the core specification.

An instance matches successfully if its primitive type is one of the types defined by keyword. Recall: "number" includes "integer".

5.22. allOf

This keyword's value MUST be an array. This array MUST have at least one element.

Elements of the array MUST be objects. Each object MUST be a valid JSON Schema.

An instance validates successfully against this keyword if it validates successfully against all schemas defined by this keyword's value.

5.23. anyOf

This keyword's value MUST be an array. This array MUST have at least one element.

Elements of the array MUST be objects. Each object MUST be a valid JSON Schema.

An instance validates successfully against this keyword if it validates successfully against at least one schema defined by this keyword's value.

5.24. oneOf

This keyword's value MUST be an array. This array MUST have at least one element.

Elements of the array MUST be objects. Each object MUST be a valid JSON Schema.

An instance validates successfully against this keyword if it validates successfully against exactly one schema defined by this keyword's value.

5.25. not

This keyword's value MUST be an object. This object MUST be a valid JSON Schema.

An instance is valid against this keyword if it fails to validate successfully against the schema defined by this keyword.

5.26. definitions

This keyword's value MUST be an object. Each member value of this object MUST be a valid JSON Schema.

This keyword plays no role in validation per se. Its role is to provide a standardized location for schema authors to inline JSON Schemas into a more general schema.

As an example, here is a schema describing an array of positive integers, where the positive integer constraint is a subschema in "definitions":

```
{
    "type": "array",
    "items": { "$ref": "#/definitions/positiveInteger" },
    "definitions": {
        "positiveInteger": {
            "type": "integer",
            "minimum": 0,
            "exclusiveMinimum": true
        }
    }
}
```

6. Metadata keywords

6.1. "title" and "description"

The value of both of these keywords MUST be a string.

Both of these keywords can be used to decorate a user interface with information about the data produced by this user interface. A title will preferrably be short, whereas a description will provide explanation about the purpose of the instance described by this schema.

Both of these keywords MAY be used in root schemas, and in any subschemas.

6.2. "default"

There are no restrictions placed on the value of this keyword.

This keyword can be used to supply a default JSON value associated with a particular schema. It is RECOMMENDED that a default value be valid against the associated schema.

This keyword MAY be used in root schemas, and in any subschemas.

7. Semantic validation with "format"

7.1. Foreword

Structural validation alone may be insufficient to validate that an instance meets all the requirements of an application. The "format" keyword is defined to allow interoperable semantic validation for a fixed subset of values which are accurately described by authoritative resources, be they RFCs or other external specifications.

The value of this keyword is called a format attribute. It MUST be a string. A format attribute can generally only validate a given set of instance types. If the type of the instance to validate is not in this set, validation for this format attribute and instance SHOULD succeed.

7.2. Implementation requirements

Implementations MAY support the "format" keyword. Should they choose to do so:

- they SHOULD implement validation for attributes defined below;
- they SHOULD offer an option to disable validation for this keyword.

Implementations MAY add custom format attributes. Save for agreement between parties, schema authors SHALL NOT expect a peer implementation to support this keyword and/or custom format attributes.

7.3. Defined formats

7.3.1. date-time

This attribute applies to string instances.

A string instance is valid against this attribute if it is a valid date representation as defined by RFC 3339, section 5.6 [RFC3339].

7.3.2. email

This attribute applies to string instances.

A string instance is valid against this attribute if it is a valid Internet email address as defined by RFC 5322, section 3.4.1 [RFC5322].

7.3.3. hostname

7.3.3.1. Applicability

This attribute applies to string instances.

7.3.3.2. Validation

A string instance is valid against this attribute if it is a valid representation for an Internet host name, as defined by RFC 1034, section 3.1 [RFC1034].

7.3.4. ipv4

This attribute applies to string instances.

A string instance is valid against this attribute if it is a valid representation of an IPv4 address according to the "dotted-quad" ABNF syntax as defined in RFC 2673, section 3.2 [RFC2673].

7.3.5. ipv6

This attribute applies to string instances.

A string instance is valid against this attribute if it is a valid representation of an IPv6 address as defined in RFC 2373, section 2.2 [RFC2373].

7.3.6. uri

This attribute applies to string instances.

A string instance is valid against this attribute if it is a valid URI, according to [RFC3986].

7.3.7. uriref

This attribute applies to string instances.

A string instance is valid against this attribute if it is a valid URI Reference (either a URI or a relative-reference), according to [RFC3986].

8. Security considerations

JSON Schema validation defines a vocabulary for JSON Schema core and conserns all the security considerations listed there.

JSON Schema validation allows the use of Regular Expressions, which have numerous different (often incompatible) implementations. Some implementations allow the embedding of arbritrary code, which is outside the scope of JSON Schema and MUST NOT be permitted. Regular expressions can often also be crafted to be extremely expensive to compute (with so-called "catastrophic backtracking"), resulting in a denial-of-service attack.

9. IANA Considerations

This specification does not have any influence with regards to IANA.

10. References

10.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[json-schema] "JSON Schema: A Media Type for Describing JSON Documents", draft-wright-json-schema-00 (work in progress), October 2016.

10.2. Informative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <http://www.rfc-editor.org/info/rfc1034>.

[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, DOI 10.17487/RFC2373, July 1998, <http://www.rfc-editor.org/info/rfc2373>.

[RFC2673] Crawford, M., "Binary Labels in the Domain Name System", RFC 2673, DOI 10.17487/RFC2673, August 1999, <http://www.rfc-editor.org/info/rfc2673>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

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

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

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <http://www.rfc-editor.org/info/rfc5322>.

[ecma262] "ECMA 262 specification", <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.

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, and Henry H. Andrews for their submissions and patches to the document.

Appendix B. ChangeLog

[[CREF1: This section to be removed before leaving Internet-Draft status.]]

draft-wright-json-schema-validation-00

* Added additional security considerations
* Removed reference to "latest version" meta-schema, use numbered version instead
* Rephrased many keyword definitions for brevity
* Added "uriref" format that also allows relative URI references

draft-fge-json-schema-validation-01

* Initial draft.
* Salvaged from draft v3.
* Redefine the "required" keyword.
* Remove "extends", "disallow"
* Add "anyOf", "allOf", "oneOf", "not", "definitions", "minProperties", "maxProperties".
* "dependencies" member values can no longer be single strings; at least one element is required in a property dependency array.
* Rename "divisibleBy" to "multipleOf".
* "type" arrays can no longer have schemas; remove "any" as a possible value.
* Rework the "format" section; make support optional.

* "format": remove attributes "phone", "style", "color"; rename "ip-address" to "ipv4"; add references for all attributes.

* Provide algorithms to calculate schema(s) for array/object instances.

* Add interoperability considerations.

Authors' Addresses

Austin Wright (editor)

EMail: aaa@bzfx.net

Geraint Luff

EMail: luffgd@gmail.com