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

JSON Schema Validation: A Vocabulary for Structural Validation of JSON

draft-wright-json-schema-validation-01

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 October 23, 2017.

Copyright Notice

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.

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 constrain values within a certain primitive type. When the 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. Validation of primitive types and child values

Two of the primitive types, array and object, allow for child values. The validation of the primitive type is considered separately from the validation of child instances.

For arrays, primitive type validation consists of validating restrictions on length with "minItems" and "maxItems", while "items" and "additionalItems" determine which subschemas apply to which elements of the array. In addition, "uniqueItems" and "contains" validate array contents as a whole.

For objects, primitive type validation consists of validating restrictions on which and how many properties appear with "required", "minProperties", "maxProperties", "propertyNames", and the string array form of "dependencies", while "properties", "patternProperties", and "additionalProperties" determine which subschemas apply to which object property values. In addition, the schema form of "dependencies" validates the object as a whole based on the presence of specific property names.

4.3. Constraints and missing keywords

Each JSON Schema validation keyword adds constraints that an instance must satisfy in order to successfully validate.

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.

4.4. Keyword independence

Validation keywords typically operate independently, without affecting each other's outcomes.

For schema author convenience, 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. Meta-schema

The current URI for the JSON Schema Validation is <http://json-schema.org/draft-06/schema#>.

6. Validation keywords

Validation keywords in a schema impose requirements for successful validation of an instance.

6.1. multipleOf

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

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

6.2. maximum

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

If the instance is a number, then this keyword validates only if the instance is less than or exactly equal to "maximum".

6.3. exclusiveMaximum

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

If the instance is a number, then the instance is valid only if it has a value strictly less than (not equal to) "exclusiveMaximum".

6.4. minimum

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

If the instance is a number, then this keyword validates only if the instance is greater than or exactly equal to "minimum".

6.5. exclusiveMinimum

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

If the instance is a number, then the instance is valid only if it has a value strictly greater than (not equal to) "exclusiveMinimum".

6.6. maxLength

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

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].

6.7. minLength

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

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].

Omitting this keyword has the same behavior as a value of 0.

6.8. 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.

6.9. items

The value of "items" MUST be either a valid JSON Schema or an array of valid JSON Schemas.

This keyword determines how child instances validate for arrays, and does not directly validate the immediate instance itself.

If "items" is a schema, validation succeeds if all elements in the array successfully validate against that schema.

If "items" is an array of schemas, validation succeeds if each element of the instance validates against the schema at the same position, if any.

Omitting this keyword has the same behavior as an empty schema.

6.10. additionalItems

The value of "additionalItems" MUST be a valid JSON Schema.

This keyword determines how child instances validate for arrays, and does not directly validate the immediate instance itself.

If "items" is an array of schemas, validation succeeds if every instance element at a position greater than the size of "items" validates against "additionalItems".

Otherwise, "additionalItems" MUST be ignored, as the "items" schema (possibly the default value of an empty schema) is applied to all elements.

Omitting this keyword has the same behavior as an empty schema.

6.11. maxItems

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

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

6.12. minItems

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

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

Omitting this keyword has the same behavior as a value of 0.

6.13. 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.

Omitting this keyword has the same behavior as a value of false.

6.14. contains

The value of this keyword MUST be a valid JSON Schema.

An array instance is valid against "contains" if at least one of its elements is valid against the given schema.

6.15. maxProperties

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

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

6.16. minProperties

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

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

Omitting this keyword has the same behavior as a value of 0.

6.17. required

The value of this keyword MUST be an array. Elements of this array, if any, MUST be strings, and MUST be unique.

An object instance is valid against this keyword if every item in the array is the name of a property in the instance.

Omitting this keyword has the same behavior as an empty array.

6.18. properties

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

This keyword determines how child instances validate for objects, and does not directly validate the immediate instance itself.

Validation succeeds if, for each name that appears in both the instance and as a name within this keyword's value, the child instance for that name successfully validates against the corresponding schema.

Omitting this keyword has the same behavior as an empty object.

6.19. 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 a valid JSON Schema.

This keyword determines how child instances validate for objects, and does not directly validate the immediate instance itself. Validation of the primitive instance type against this keyword always succeeds.

Validation succeeds if, for each instance name that matches any regular expressions that appear as a property name in this keyword's value, the child instance for that name successfully validates against each schema that corresponds to a matching regular expression.

Omitting this keyword has the same behavior as an empty object.

6.20. additionalProperties

The value of "additionalProperties" MUST be a valid JSON Schema.

This keyword determines how child instances validate for objects, and does not directly validate the immediate instance itself.

Validation with "additionalProperties" applies only to the child values of instance names that do not match any names in "properties", and do not match any regular expression in "patternProperties".

For all such properties, validation succeeds if the child instance validates against the "additionalProperties" schema.

Omitting this keyword has the same behavior as an empty schema.

6.21. 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 array or a valid JSON Schema.

If the dependency value is a subschema, and the dependency key is a property in the instance, the entire instance must validate against the dependency value.

If the dependency value is an array, each element in the array, if any, MUST be a string, and 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.

Omitting this keyword has the same behavior as an empty object.

6.22. propertyNames

The value of "propertyNames" MUST be a valid JSON Schema.

If the instance is an object, this keyword validates if every property name in the instance validates against the provided schema. Note the property name that the schema is testing will always be a string.

Omitting this keyword has the same behavior as an empty schema.

6.23. 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.

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

Elements in the array might be of any value, including null.

6.24. const

The value of this keyword MAY be of any type, including null.

An instance validates successfully against this keyword if its value is equal to the value of the keyword.

6.25. 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 six primitive types ("null", "boolean", "object", "array", "number", or "string"), or "integer" which matches any number with a zero fractional part.

An instance validates if and only if the instance is in any of the sets listed for this keyword.

6.26. allOf

This keyword's value MUST be a non-empty array. Each item of the array 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.

6.27. anyOf

This keyword's value MUST be a non-empty array. Each item of the array 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.

6.28. oneOf

This keyword's value MUST be a non-empty array. Each item of the array 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.

6.29. not

This keyword's value 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.

7. Metadata keywords

7.1. 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.

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

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

7.2. "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 preferably be short, whereas a description will provide explanation about the purpose of the instance described by this schema.

7.3. "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.

7.4. "examples"

The value of this keyword MUST be an array. There are no restrictions placed on the values within the array.

This keyword can be used to provide sample JSON values associated with a particular schema, for the purpose of illustrating usage. It is RECOMMENDED that these values be valid against the associated schema.

Implementations MAY use the value of "default", if present, as an additional example. If "examples" is absent, "default" MAY still be used in this manner.

8. Semantic validation with "format"

8.1. 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.

8.2. 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.

8.3. Defined formats

8.3.1. 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].

8.3.2. 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].

8.3.3. hostname

This attribute applies to string instances.

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].

8.3.4. 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].

8.3.5. 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].

8.3.6. uri

This attribute applies to string instances.

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

8.3.7. uri-reference

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].

8.3.8. uri-template

This attribute applies to string instances.

A string instance is valid against this attribute if it is a valid URI Template (of any level), according to [RFC6570].

8.3.9. json-pointer

This attribute applies to string instances.

A string instance is valid against this attribute if it is a valid JSON Pointer, according to [RFC6901]

9. Security considerations

JSON Schema validation defines a vocabulary for JSON Schema core and concerns 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 arbitrary 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.

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.

[json-schema] JSON Schema: A Media Type for Describing JSON Documents", Internet-Draft draft-wright-json-schema-00, October 2016.

10.2. Informative References

, "

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987.

[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, DOI 10.17487/RFC2373, July 1998.

[RFC2673] Crawford, M., "Binary Labels in the Domain Name System", RFC 2673, DOI 10.17487/RFC2673, August 1999.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002.

[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.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M. and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012.

[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.

[RFC5322] Resnick, P., Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008.

[ecma262] ECMA 262 specification"

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

[CREF1]

draft-wright-json-schema-validation-01

- Standardized on hyphenated format names ("uriref" becomes "uri-ref")
- Add the formats "uri-template" and "json-pointer"
- Changed "exclusiveMaximum"/"exclusiveMinimum" from boolean modifiers of "maximum"/"minimum" to independent numeric fields.
- Split the additionalItems/items into two sections
- Reworked properties/patternProperties/additionalProperties definition
- Added "examples" keyword
- Added "contains" keyword
- Allow empty "required" and "dependencies" arrays
- Fixed "type" reference to primitive types
- Added "const" keyword
- Added "propertyNames" keyword

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

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