Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: August 5, 2013

fge. Galiegue, Ed.

K. Zyp
SitePen (USA)
G. Court
February 1, 2013

JSON Schema: interactive and non interactive validation
draft-fge-json-schema-validation-00

Abstract

JSON Schema (application/schema+json) has several purposes, one of which is instance validation. The validation process may be interactive or non interactive. For instance, applications may use JSON Schema to build a user interface enabling interactive content generation in addition to user input checking, or validate data retrieved from various sources. This specification describes schema keywords dedicated to validation purposes.

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 materialized by a set of keywords which are described in this specification. In addition, a set of keywords is defined to assist in interactive instance generation. Those are also described in this specification.

This specification will use the terminology defined by the JSON Schema core specification. It is advised that readers have a copy of this 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, as defined by the core specification.

3. Interoperability considerations

3.1. Validation of string instances

It should be noted that the nul character (\x00) 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 does not define any bounds to the scale or precision of numeric values. JSON Schema does not define any such bounds either. This means that numeric instances processed by JSON Schema can be arbitrarily large and/or have an arbitrarily large 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 [RFC4627];
- 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 consider that regular expressions are anchored, neither at the beginning nor at the end. This means, for instance, that "es" matches "expression".

4. General validation considerations

4.1. Keywords and instance primitive types

Some validation keywords only apply to one or more primitive types. When the primitive type of the instance cannot be validated by a given keyword, validation for this keyword and instance SHOULD succeed.

This specification groups keywords in different sections, according to the primitive type, or types, these keywords validate. Note that some keywords validate all instance types.

4.2. Inter-dependent keywords

In order to validate an instance, some keywords are influenced by the presence (or absence) of other keywords. In this case, all these keywords will be grouped in the same section.

4.3. Default values for missing keywords

Some keywords, if absent, MAY be regarded by implementations as having a default value. In this case, the default value will be mentioned.

4.4. Validation of container instances

Keywords with the possibility to validate container instances (arrays or objects) only validate the instances themselves and not their children (array items or object properties). Some of these keywords do, however, contain information which is necessary for calculating which schema(s) a child must be valid against. The algorithms to calculate a child instance's relevant schema(s) are explained in a separate section.

It should be noted that while an array element will only have to validate against one schema, object member values may have to validate against more than one schema.

5. Validation keywords sorted by instance types

5.1. Validation keywords for numeric instances (number and integer)

5.1.1. multipleOf

5.1.1.1. Valid values

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

5.1.1.2. Conditions for successful validation

A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.

5.1.2. maximum and exclusiveMaximum

5.1.2.1. Valid values

The value of "maximum" MUST be a JSON number. The value of "exclusiveMaximum" MUST be a boolean.

If "exclusiveMaximum" is present, "maximum" MUST also be present.

5.1.2.2. Conditions for successful validation

Successful validation depends on the presence and value of "exclusiveMaximum":

if "exclusiveMaximum" is not present, or has boolean value false, then the instance is valid if it is lower than, or equal to, the value of "maximum";

if "exclusiveMaximum" has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".

5.1.2.3. Default value

"exclusiveMaximum", if absent, may be considered as being present with boolean value false.

5.1.3. minimum and exclusiveMinimum

5.1.3.1. Valid values

The value of "minimum" MUST be a JSON number. The value of "exclusiveMinimum" MUST be a boolean.

If "exclusiveMinimum" is present, "minimum" MUST also be present.

5.1.3.2. Conditions for successful validation

Successful validation depends on the presence and value of "exclusiveMinimum":

if "exclusiveMinimum" is not present, or has boolean value false, then the instance is valid if it is greater than, or equal to, the value of "minimum";

if "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".

5.1.3.3. Default value

"exclusiveMinimum", if absent, may be considered as being present with boolean value false.

5.2. Validation keywords for strings

5.2.1. maxLength

5.2.1.1. Valid values

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

5.2.1.2. Conditions for successful validation

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 4627 [RFC4627].

5.2.2. minLength

5.2.2.1. Valid values

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

5.2.2.2. Conditions for successful validation

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 4627 [RFC4627].

5.2.2.3. Default value

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

5.2.3. pattern

5.2.3.1. Valid values

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.

5.2.3.2. Conditions for successful validation

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

5.3. Validation keywords for arrays

5.3.1. additionalItems and items

5.3.1.1. Valid values

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 an object or an array. If it is an object, this object MUST be a valid JSON Schema. If it is an array, items of this array MUST be objects, and each of these objects MUST be a valid JSON Schema.

5.3.1.2. Conditions for successful validation

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".

5.3.1.3. Example

The following example covers the case where "additionalItems" has boolean value false and "items" is an array, since this is the only situation under which an instance may fail to validate successfully.

This is an example schema:

```
{
    "items": [ {}, {}, {} ],
    "additionalItems": false
```

```
}
```

With this schema, the following instances are valid:

```
[] (an empty array),

[ [ 1, 2, 3, 4 ], [ 5, 6, 7, 8 ] ],

[ 1, 2, 3 ];
```

the following instances are invalid:

```
[ 1, 2, 3, 4 ],

[ null, { "a": "b" }, true, 31.000002020013 ]
```

5.3.1.4. Default values

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

5.3.2. maxItems

5.3.2.1. Valid values

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

5.3.2.2. Conditions for successful validation

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

5.3.3. minItems

5.3.3.1. Valid values

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

5.3.3.2. Conditions for successful validation

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

5.3.3.3. Default value

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

5.3.4. uniqueItems

5.3.4.1. Valid values

The value of this keyword MUST be a boolean.

5.3.4.2. Conditions for successful validation

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.

5.3.4.3. Default value

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

5.4. Validation keywords for objects

5.4.1. maxProperties

5.4.1.1. Valid values

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

5.4.1.2. Conditions for successful validation

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

5.4.2. minProperties

5.4.2.1. Valid values

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

5.4.2.2. Conditions for successful validation

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

5.4.2.3. Default value

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

5.4.3. required

5.4.3.1. Valid values

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.

5.4.3.2. Conditions for successful validation

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

5.4.4. additionalProperties, properties and patternProperties

5.4.4.1. Valid values

The value of "additionalProperties" MUST be a boolean or an object. If it is an object, it MUST also be a valid JSON Schema.

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.

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.

5.4.4.2. Conditions for successful validation

Successful validation of an object instance against these three keywords depends on the value of "additionalProperties":

if its value is boolean true or a schema, validation succeeds;

if its value is boolean false, the algorithm to determine validation success is described below.

5.4.4.3. Default values

If either "properties" or "patternProperties" are absent, they can be considered present with an empty object as a value.

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

5.4.4.4. If "additionalProperties" has boolean value false

In this case, validation of the instance depends on the property set of "properties" and "patternProperties". In this section, the property names of "patternProperties" will be called regexes for convenience.

The first step is to collect the following sets:

s The property set of the instance to validate.

p The property set from "properties".

pp The property set from "patternProperties".

Having collected these three sets, the process is as follows:

remove from "s" all elements of "p", if any;

for each regex in "pp", remove all elements of "s" which this regex matches.

Validation of the instance succeeds if, after these two steps, set "s" is empty.

5.4.4.5. Example

This schema will be used as an example:

```
{
    "properties": {
        "p1": {}
    },
    "patternProperties": {
        "p": {},
        "[0-9]": {}
    },
    "additionalProperties": false
}
```

This is the instance to validate:

```
{
    "p1": true,
    "p2": null,
    "a32&o": "foobar",
    "": [],
    "fiddle": 42,
    "apple": "pie"
}
```

The three property sets are:

s ["p1", "p2", "a32&o", "", "fiddle", "apple"]

p ["p1"]

pp ["p", "[0-9]"]

Applying the two steps of the algorithm:

after the first step, "p1" is removed from "s";

after the second step, "p2" (matched by "p"), "a32&o" (matched by "[0-9]") and "apple" (matched by "p") are removed from "s".

The set "s" still contains two elements, "" and "fiddle". Validation therefore fails.

5.4.5. dependencies

5.4.5.1. Valid values

This keyword's value MUST be an object. Each value of this object MUST be either an object or an array.

If the value is an object, it MUST be a valid JSON Schema. This is called a schema dependency.

If the 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. This is called a property dependency.

5.4.5.2. Conditions for successful validation

5.4.5.2.1. Schema dependencies

For all (name, schema) pair of schema dependencies, if the instance has a property by this name, then it must also validate successfully against the schema.

Note that this is the instance itself which must validate successfully, not the value associated with the property name.

5.4.5.2.2. Property dependencies

For each (name, propertyset) pair of property dependencies, if the instance has a property by this name, then it must also have properties with the same names as propertyset.

5.5. Validation keywords for any instance type

5.5.1. enum

5.5.1.1. Valid values

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

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

5.5.1.2. Conditions for successful validation

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

5.5.2. type

5.5.2.1. Valid values

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.

5.5.2.2. Conditions for successful validation

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

5.5.3. allOf

5.5.3.1. Valid values

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.

5.5.3.2. Conditions for successful validation

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

5.5.4. anyOf

5.5.4.1. Valid values

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.

5.5.4.2. Conditions for successful validation

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

5.5.5. oneOf

5.5.5.1. Valid values

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.

5.5.5.2. Conditions for successful validation

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

5.5.6. not

5.5.6.1. Valid values

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

5.5.6.2. Conditions for successful validation

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

5.5.7. definitions

5.5.7.1. Valid values

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

5.5.7.2. Conditions for successful validation

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"

6.1.1. Valid values

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

6.1.2. Purpose

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"

6.2.1. Valid values

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

6.2.2. Purpose

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 attributes

7.3.1. date-time

7.3.1.1. Applicability

This attribute applies to string instances.

7.3.1.2. Validation

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

7.3.2.1. Applicability

This attribute applies to string instances.

7.3.2.2. Validation

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

7.3.4.1. Applicability

This attribute applies to string instances.

7.3.4.2. Validation

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

7.3.5.1. Applicability

This attribute applies to string instances.

7.3.5.2. Validation

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

7.3.6.1. Applicability

This attribute applies to string instances.

7.3.6.2. Validation

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

8. Reference algorithms for calculating children schemas

8.1. Foreword

Calculating the schema, or schemas, a child instance must validate against is influenced by the following:

the container instance type;

the child instance's defining characteristic in the container instance;

the value of keywords implied in the calculation.

In addition, it is important that if one or more keyword(s) implied in the calculation are not defined, they be considered present with their default value, which will be recalled in this section.

8.2. Array elements

8.2.1. Defining characteristic

The defining characteristic of the child instance is its index within the array. Recall: array indices start at 0.

8.2.2. Implied keywords and default values.

The two implied keywords in this calculation are "items" and "additionalItems".

If either keyword is absent, it is considered present with an empty schema as a value. In addition, boolean value true for "additionalItems" is considered equivalent to an empty schema.

8.2.3. Calculation

8.2.3.1. If "items" is a schema

If items is a schema, then the child instance must be valid against this schema, regardless of its index, and regardless of the value of "additionalItems".

8.2.3.2. If "items" is an array

In this situation, the schema depends on the index:

if the index is less than, or equal to, the size of "items", the child instance must be valid against the corresponding schema in the "items" array;

otherwise, it must be valid against the schema defined by "additionalItems".

8.3. Object members

8.3.1. Defining characteristic

The defining characteristic is the property name of the child.

8.3.2. Implied keywords

The three keywords implied in this calculation are "properties", "patternProperties" and "additionalProperties".

If "properties" or "patternProperties" are absent, they are considered present with an empty object as a value.

If "additionalProperties" is absent, it is considered present with an empty schema as a value. In addition, boolean value true is considered equivalent to an empty schema.

8.3.3. Calculation

8.3.3.1. Names used in this calculation

The calculation below uses the following names:

m The property name of the child.

p The property set from "properties".

pp The property set from "patternProperties". Elements of this set will be called regexes for convenience.

s The set of schemas for the child instance.

8.3.3.2. First step: schemas in "properties"

If set "p" contains value "m", then the corresponding schema in "properties" is added to "s".

8.3.3.3. Second step: schemas in "patternProperties"

For each regex in "pp", if it matches "m" successfully, the corresponding schema in "patternProperties" is added to "s".

8.3.3.4. Third step: "additionalProperties"

The schema defined by "additionalProperties" is added to "s" if and only if, at this stage, "s" is empty.

9. Security considerations

JSON Schema validation does not have any additional security considerations than those defined by the JSON Schema core specification.

10. IANA Considerations

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

11. References

11.1. Normative References

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

11.2. Informative References

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

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

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

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

[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.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

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

Appendix A. ChangeLog

draft-00

* 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

Francis Galiegue (editor)

EMail: fgaliegue@gmail.com

Kris Zyp
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