Building Blocks for HTTP APIs
Internet-Draft
Intended status: Standards Track
Expires: 3 January 2026

C. Vasters
Microsoft Corporation

2 July 2025

JSON Structure: Validation Extensions
draft-vasters-json-structure-validation-00

Abstract

The JSON Structure Validation extension provides schema authors with additional means to constrain instance data. These keywords are applied in conjunction with the constructs defined in JSON Structure Core. The keywords defined herein include numeric, string, array, and object validation keywords as well as conditional validations.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://json-structure.github.io/validation/draft-vasters-json-structure-validation.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-vasters-json-structure-validation/.

Source for this draft and an issue tracker can be found at https://github.com/json-structure/validation.

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 https://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 3 January 2026.

Table of Contents

1. Introduction

The JSON Structure Validation extension provides schema authors with additional means to constrain instance data. These keywords are applied in conjunction with the constructs defined in JSON Structure Core [JSTRUCT-CORE]. The keywords defined herein include numeric, string, array, and object validation keywords as well as conditional validations.

For each keyword, this document specifies its applicability, the permitted value types, and the related standards that must be observed.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Validation Keywords

3.1. Numeric Validation Keywords

This section defines the validation keywords applicable to schemas with numeric types. The value of each keyword MUST be in the value space of the numeric type to which the keyword is applied.

For schemas with extended numeric types (such as long integers and decimals) whose base representation is a string, numeric constraint values (e.g., for minimum, maximum) MUST be provided as strings.

3.1.1. minimum

An instance is valid if its numeric value is greater than or equal to the value specified in minimum.

Example for basic type:

```
{ "type": "number", "minimum": 10 }
```

Example for extended type:

```
{ "type": "decimal", "minimum": "10.00" }
```

3.1.2. maximum

An instance is valid if its numeric value is less than or equal to the value specified in maximum.

Example for basic type:

```
{ "type": "number", "maximum": 100 }
```

Example for extended type:

```
{ "type": "decimal", "maximum": "100.00" }
```

3.1.3. exclusiveMinimum

An instance is valid if its numeric value is strictly greater than the value specified in exclusiveMinimum.

Example for basic type:

```
{ "type": "number", "exclusiveMinimum": 10 }
```

Example for extended type:

```
{ "type": "int64", "exclusiveMinimum": "10" }
```

3.1.4. exclusiveMaximum

An instance is valid if its numeric value is strictly less than the value specified in exclusiveMaximum.

Example for basic type:

```
{ "type": "number", "exclusiveMaximum": 100 }
```

Example for extended type:

```
{ "type": "decimal", "exclusiveMaximum": "100.00" }
```

3.1.5. multipleOf

An instance is valid if dividing its numeric value by the value of multipleOf results in an integer value. The value of multipleOf MUST be a positive number.

Example for basic type:

```
{ "type": "number", "multipleOf": 5 }
```

Example for extended type:

```
{ "type": "decimal", "multipleOf": "5" }
```

3.2. String Validation Keywords

This section defines the validation keywords applicable to schemas with the type string. The maxLength keyword is not included as it is part of JSON Structure Core and is not redefined here.

3.2.1. minLength

A string is valid if its length is at least the integer value specified in minLength. The value of minLength MUST be a non-negative integer.

Example:

```
{ "type": "string", "minLength": 3 }
```

3.2.2. pattern

A string is valid if its entire value conforms to the regular expression provided in the pattern keyword. The value of pattern MUST be a string representing a valid regular expression that conforms to the [ECMA_262_2022] standard.

Example:

```
{ "type": "string", "pattern": "^[A-Z][a-z]+$" }
```

3.2.3. format

A string is valid if it conforms to a specific format. The value of format MUST be a string. The format keyword adds additional standard validation constraints not covered by the extended types in the core specification without the need to define an explicit regular expression pattern.

* ipv4 Internet Protocol version 4 address
* ipv6 Internet Protocol version 6 address
* email Email address
* idn-email Internationalized email address

* hostname Hostname
* idn-hostname Internationalized hostname
* iri Internationalized Resource Identifier
* iri-reference Internationalized Resource Identifier reference
* uri-template URI template
* relative-json-pointer Relative JSON pointer
* regex Regular expression

3.3. Array and Set Validation Keywords

This section defines the validation keywords applicable to schemas with the type array and set.

3.3.1. minItems

An array or set is valid if its number of elements is at least the integer value specified in minItems. The value of minItems MUST be a non-negative integer.

Example:

```
{ "type": "array", "minItems": 2 }
```

3.3.2. maxItems

An array or set is valid if its number of elements does not exceed the integer value specified in maxItems. The value of maxItems MUST be a non-negative integer.

Example:

```
{ "type": "array", "maxItems": 10 }
```

3.3.3. uniqueItems

This keyword is only applicable to schemas with the type array as this constraint is inherent to set. An array is valid if, when uniqueItems is set to true, no two elements are equal. The value of uniqueItems MUST be a boolean (either true or false).

Example:

```
{ "type": "array", "uniqueItems": true }
```

3.3.4. contains

An array or set is valid if at least one element satisfies the schema specified in contains. The value of contains MUST be a valid JSON Structure object.

Example:

```
{ "type": "array", "contains": { "type": "string" } }
```

The condition schema MAY contain a const keyword to specify a fixed value that the array must contain.

Example:

```
{ "type": "array", "contains": { "type": "string", "const": "foo" } }
```

3.3.5. maxContains

An array or set is valid if at most the number of elements specified in maxContains satisfy the schema specified in contains. The value of maxContains MUST be a non-negative integer.

Example:

```
{ "type": "array", "contains": { "type": "string" }, "maxContains": 2 }
```

3.3.6. minContains

An array or set is valid if at least the number of elements specified in minContains satisfy the schema specified in contains. The value of minContains MUST be a non-negative integer.

Example:

```
{ "type": "array", "contains": { "type": "string" }, "minContains": 2 }
```

3.4. Object and Map Validation Keywords

This section defines the validation keywords applicable to schemas with the type object and map.

3.4.1. minProperties and minEntries

An object is valid if it has at least as many properties as defined by the integer value specified in minProperties. The value of minProperties MUST be a non-negative integer. The minEntries keyword applies equivalently to map types.

Example:

```
{ "type": "object", "minProperties": 1 }
```

This constraint is useful for object definitions that use dynamic properties via additionalProperties and patternProperties.

3.4.2. maxProperties and maxEntries

An object is valid if it contains no more than the integer value specified in maxProperties. The value of maxProperties MUST be a non-negative integer. The maxEntries keyword applies equivalently to map types.

Example:

```
{ "type": "object", "maxProperties": 5 }
```

3.4.3. dependentRequired

This keyword establishes dependencies between object properties. The value is a map of arrays of strings. Each entry in the map corresponds to a property name in the object instance. If the property exists, then the properties listed in the corresponding array MUST also exist in the instance. This keyword does not apply to the map type.

Example:

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "credit_card": { "type": "number" },
    "billing_address": { "type": "string" }
  },
  "dependentRequired": {
    "credit_card": ["billing_address"]
  },
  "required": ["name"]
}
```

3.4.4. patternProperties and patternKeys

This keyword applies schemas to properties whose names match specified regular expressions. For each property in the object instance, if its name matches a regular expression defined in patternProperties, then its value MUST validate against the corresponding schema. The property names used as keys in patternProperties MUST be strings representing valid regular expressions conforming to the [ECMA_262_2022] standard. The patternKeys keyword applies equivalently to map types.

Example:

```
{
  "type": "object",
  "patternProperties": {
    "^[A-Z]": { "type": "string" }
  }
}
```

Note: All identifiers are additionally subject to the constraints of the identifier syntax in JSON Structure Core [JSTRUCT-CORE].

3.4.5. propertyNames and keyNames

The propertyNames keyword validates the names of all properties in an object against a string-typed schema. An object is valid if every property name in the object is valid. The schema MUST be of type string. The keyNames keyword applies equivalently to map types.

Example:

```
{
  "type": "object",
  "propertyNames": { "type": "string", "pattern": "^[a-z][a-zA-Z0-9]*$" }
}
```

3.4.6. has

The has keyword validates that an object or map has at least one (property) value that matches the schema. The schema MUST be of type object.

Example:

```
{
  "type": "object",
  "has": { "type": "string" }
}
```

3.5. Default Values

3.5.1. default

The default keyword provides a default value for a schema. If an instance matches the schema but does not contain the property, the default value is used.

Example:

```
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "default": "John Doe"
    }
  }
}
```

3.6. Enabling the Extnensions

Validation extensions can be enabled in a schema or meta-schema by adding the JSONSchemaValidation key to the $uses clause when referencing the extended meta-schema:

```
{
  "$schema": "https://json-structure.org/meta/extended/v0/#",
  "$id": "myschema",
  "$uses": [
    "JSONSchemaValidation",
  ],
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "pattern": "^[A-Z][a-z]+$"
    }
  }
}
```

The extensions are enabled by default in the validation meta-schema:

```
{
  "$schema": "https://json-structure.org/meta/validation/v0/#",
  "$id": "myschema",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "pattern": "^[A-Z][a-z]+$"
    }
  }
}
```

4. Implementation Considerations

Validators shall process each validation keyword independently and combine results using a logical AND conjunction. Regular expression evaluation for pattern, patternProperties, and propertyNames MUST conform to the ECMAScript Language Specification [ECMA_262_2022].

5. Security Considerations

Complex regular expressions specified in pattern, patternProperties, and propertyNames may lead to performance issues (e.g., ReDoS). Implementations should mitigate such risks. Overly complex or deeply nested validation constructs may impact performance and should be optimized.

6. IANA Considerations

This document has no IANA actions.

7. Normative References

[ECMA_262_2022]
Ecma International, "ECMAScript Language Specification", ECMA Standards ECMA-262, 2022, <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>.

[JSTRUCT-CORE]
Vasters, C., "JSON Structure Core", n.d., <https://json-structure.github.io/core/draft-vasters-json-structure-core.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC4646] Phillips, A. and M. Davis, "Tags for Identifying Languages", RFC 4646, DOI 10.17487/RFC4646, September 2006, <https://www.rfc-editor.org/rfc/rfc4646>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

Acknowledgments

TODO acknowledge.

Author's Address

Clemens Vasters
Microsoft Corporation
Email: clemensv@microsoft.com