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

C. Vasters
Microsoft Corporation

2 July 2025

JSON Structure: Conditional Composition
draft-vasters-json-structure-cond-composition-00

Abstract

This document specifies JSON Structure Conditional Composition, an extension to JSON Structure Core that introduces composition constructs for combining multiple schema definitions. In particular, this specification defines the semantics, syntax, and constraints for the keywords allOf, anyOf, oneOf, and not, as well as the if/then/else conditional construct.

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/conditional-composition/draft-vasters-json-structure-cond-composition.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-vasters-json-structure-cond-composition/.

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

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

This document specifies JSON Structure Conditionals, an extension to JSON Structure Core [JSTRUCT-CORE] that introduces conditional composition constructs for combining multiple schema definitions. In particular, this specification defines the semantics, syntax, and constraints for the keywords allOf, anyOf, oneOf, and not, as well as the if/then/else conditional construct.

2. Terminology and Conventions

The key words MUST, MUST NOT, SHALL, SHALL NOT, REQUIRED, SHOULD, and OPTIONAL are to be interpreted as described in [RFC2119] and [RFC8174].

Unless otherwise specified, all references to "non-schema" refer to a JSON Structure Core non-schema object, which is an inert JSON object that does not declare a type constraint.

3. Composition and Evaluation Model

The keywords introduced in this document extend the set of keywords allowed for non-schemas and schemas as defined in JSON Structure Core. In particular, the keywords defined herein MAY extend all non-schema and schema definitions.

The focus of JSON Structure Core is on data definitions. The conditional composition keywords introduced in this document allow authors to define conditional matching rules that use these fundamental data definitions.

A schema document using these keywords is not a data definition but a rule set for evaluating JSON node instances against schema definitions and lays the groundwork for validation.

Fundamentally, evaluating a JSON node against a schema involves matching the node against the schema’s constraints.

The outcome of evaluating a JSON node against a schema is ultimately a boolean value that states whether the node met all constraints defined in the schema. The evaluation also creates an understanding of which constraint was met for each subschema during evaluation.

A schema evaluation engine traverses the given JSON node and the schema definition, evaluating the node and the schema recursively. When a conditional composition keyword is encountered, the engine evaluates each subschema independently against the current node and then combines the results as specified by the composition keyword.

4. Conditional Composition Keywords

This section defines several composition keywords that combine schema definitions with evaluation rules. Each keyword has a specific evaluation semantics that determines the outcome of the validation process.

4.1. allOf

The value of the allOf keyword MUST be a type-union array containing at least one schema object. A JSON node is valid against allOf if and only if it is valid against every schema in the array.

Consider the following non-schema, which does not define its own type but rather contains an allOf keyword with three subschemas:

```
{
  "allOf": [
    {
      "type": "object",
      "properties": {
        "a": { "type": "string" }
      },
      "required": ["a"],
      "additionalProperties": true
    },
    {
      "type": "object",
      "properties": {
        "b": { "type": "number" }
      },
      "required": ["b"],
      "additionalProperties": true
    },
    {
      "type": "object",
      "properties": {
        "c": { "type": "boolean" }
      },
      "required": ["c"],
      "additionalProperties": true
    }
  ]
}
```

Here, a JSON node evaluates to true if it is an object with at least three properties a, b, and c, where a is a string, b is a number, and c is a boolean:

```
{
  "a": "string",
  "b": 42,
  "c": true
}
```

The JSON node satisfies all constraints defined by all subschemas. Conflicting constraints among subschemas result in an unsatisfiable schema— for example, if two subschemas require the same property to have different types or if one of the subschemas has additionalProperties set to false.

4.2. anyOf

The value of the anyOf keyword MUST be a type-union array containing at least one schema object. A JSON node is valid against anyOf if and only if it is valid against at least one of the schemas in the array.

Consider the following schema:

```
{
  "anyOf": [
    {
      "type": "object",
      "properties": {
        "a": { "type": "string" }
      },
      "required": ["a"],
      "additionalProperties": true
    },
    {
      "type": "object",
      "properties": {
        "b": { "type": "number" }
      },
      "required": ["b"],
      "additionalProperties": true
    },
    {
      "type": "object",
      "properties": {
        "c": { "type": "boolean" }
      },
      "required": ["c"],
      "additionalProperties": true
    }
  ]
}
```

Here, a JSON node evaluates to true if it is an object with at least one of the properties a, b, or c, where a is a string, b is a number, and c is a boolean:

```
{
  "a": "string"
}
```

or

```
{
  "b": 42,
  "c": true
}
```

Both JSON nodes satisfy the constraints defined by at least one subschema.

4.3. oneOf

The value of the oneOf keyword MUST be a type-union array containing at least one schema object. A JSON node is valid against oneOf if and only if it is valid against exactly one of the schemas in the array.

Consider the following schema:

```
{
  "oneOf": [
    {
      "type": "object",
      "properties": {
        "a": { "type": "string" }
      },
      "required": ["a"],
      "additionalProperties": true
    },
    {
      "type": "object",
      "properties": {
        "b": { "type": "number" }
      },
      "required": ["b"],
      "additionalProperties": true
    },
    {
      "type": "object",
      "properties": {
        "c": { "type": "boolean" }
      },
      "required": ["c"],
      "additionalProperties": true
    }
  ]
}
```

Here, a JSON node evaluates to true if it is an object with exactly one of the properties a, b, or c, where a is a string, b is a number, and c is a boolean:

```
{
  "a": "string"
}
```

The following JSON node evaluates to false because it matches two subschemas:

```
{
  "a": "string",
  "b": 42
}
```

4.4. not

The value of the keyword not is a single schema object, which MAY be a type union. A JSON node is valid against not if it is not valid against the schema. For example, the schema is written as follows:

```
{
  "not": { "type": "string" }
}
```

Here, a JSON node evaluates to true if it is not a string:

```
42
```

4.5. if/then/else

The values of the keywords if, then, and else are schema objects. If the processed JSON node is valid against the if schema, the then schema further constrains the JSON node and MUST match the input. If the processed JSON node is not valid against the if schema, the else schema further constrains the JSON node and MUST match the input.

Consider the following schema:

```
{
  "if": {
    "properties": {
      "a": { "type": "string" }
    },
    "required": ["a"]
  },
  "then": {
    "properties": {
      "b": { "type": "number" }
    },
    "required": ["b"]
  },
  "else": {
    "properties": {
      "c": { "type": "boolean" }
    },
    "required": ["c"]
  }
}
```

Here, a JSON node evaluates to true if it is an object with a property a that is a string; then it must also have a property b that is a number:

```
{
  "a": "string",
  "b": 42
}
```

Otherwise, if the JSON node does not have a property a that is a string, it must have a property c that is a boolean:

```
{
  "c": true
}
```

or

```
{
  "a": 42,
  "c": false
}
```

4.6. Enabling the Extensions

The conditional composition extensions can be enabled in a schema or meta-schema by adding the JSONSchemaConditionalComposition key to the $uses clause when referencing the extended meta-schema:

```
{
  "$schema": "https://json-structure.org/meta/extended/v0/#",
  "$id": "myschema",
  "$uses": [
    "JSONSchemaConditionalComposition"
  ],
  "oneOf" : [
    { "type": "string" },
    { "type": "number" }
  ]
}
```

Conditional composition is enabled by default in the validation meta-schema:

```
{
  "$schema": "https://json-structure.org/meta/validation/v0/#",
  "$id": "myschema",
  "type": "object",
  "oneOf" : [
    { "type": "string" },
    { "type": "number" }
  ]
}
```

5. Security Considerations

* The use of composition keywords does not alter the security model of JSON Structure Core; however, excessive nesting or overly complex compositions may impact performance and resource usage.

* Implementations MUST ensure that all subschema references resolve within the same document or trusted sources to prevent external schema injection.

6. IANA Considerations

This document does not require any IANA actions.

7. Normative References

[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