Workgroup: HTTPAPI
Internet-Draft:
draft-ietf-httpapi-rest-api-mediatypes-01
Published: 7 March 2022
Intended Status: Informational
Expires: 8 September 2022
Authors: R. Polli
Digital Transformation Department, Italian Government

REST API Media Types

Abstract

This document registers the following media types used in APIs on the IANA Media Types registry: application/yaml, application/schema+json, application/schema-instance+json, application/openapi+json, and application/openapi+yaml.

Note to Readers

RFC EDITOR: please remove this section before publication

Discussion of this draft takes place on the HTTP APIs working group mailing list (httpapi@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/httpapi/.

The source code and issues list for this draft can be found at https://github.com/ietf-wg-httpapi/mediatypes.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

OpenAPI Specification [oas] version 3 and above is a consolidated standard for describing HTTP APIs using the JSON [JSON] and YAML [YAML] data format.

To increase interoperability when processing API specifications and leverage content negotiation mechanisms when exchanging OpenAPI Specification resources this specification register the following media-types: application/yaml, application/schema+json, application/schema-instance+json, application/openapi+json and application/openapi+yaml.

Moreover it defines and registers the +yaml structured syntax suffix.

1.1. Notational 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. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

This document uses the Augmented BNF defined in [RFC5234] and updated by [RFC7405].

The terms "content", "content negotiation", "resource", and "user agent" in this document are to be interpreted as in [SEMANTICS].

2. Media Type registrations

This section describes the information required to register the above media types according to [MEDIATYPE]

2.1. Media Type application/yaml

The following information serves as the registration form for the application/yaml media type.

Type name: application

Subtype name: yaml

Required parameters: None

Optional parameters: None; unrecognized parameters should be ignored

Encoding considerations: Same as [JSON]

Security considerations: see Section 4 of this document

Interoperability considerations: see Section 3.1 of this document

Published specification: (this document)

Applications that use this media type: HTTP

Fragment identifier considerations: Same as for application/json [JSON]

Additional information:

Deprecated alias names for this type: application/x-yaml, text/yaml, text/x-yaml

Magic number(s): n/a

File extension(s): yaml, yml

Macintosh file type code(s): n/a

Person and email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: None.

Author: See Authors' Addresses section.

Change controller: n/a

2.2. The +yaml Structured Syntax Suffix

The suffix +yaml MAY be used with any media type whose representation follows that established for application/yaml. The media type structured syntax suffix registration form follows. See [MEDIATYPE] for definitions of each of the registration form headings.

Name: YAML Ain't Markup LanguageML (YAML)

+suffix: +yaml

References: [YAML]

Encoding considerations: see Section 2.1

Fragment identifier considerations:

The syntax and semantics of fragment identifiers specified for +yaml SHOULD be as specified for {{application-yaml}}

The syntax and semantics for fragment identifiers for a specific `xxx/yyy+json` SHOULD be processed as follows:

For cases defined in +yaml, where the fragment identifier resolves per the +yaml rules, then process as specified in +yaml.

For cases defined in +yaml, where the fragment identifier does not resolve per the +yaml rules, then process as specified in `xxx/yyy+yaml`.

For cases not defined in +yaml, then process as specified in `xxx/yyy+yaml`.

Interoperability considerations: See Section 2.1

Security considerations: See Section 2.1

Contact: See Authors' Addresses section.

Author: See Authors' Addresses section

Change controller: n/a

2.3. The OpenAPI Media Types

The OpenAPI Specification Media Types convey OpenAPI document (OAS) files as defined in [oas] for version 3.0.0 and above.

Those files can be serialized in [JSON] or [YAML]. Since there are multiple OpenAPI Specification versions, those media-types support the version parameter.

The following examples conveys the desire of a client to receive an OpenAPI Specification resource preferably in the following order:

1. openapi 3.1 in YAML

2. openapi 3.0 in YAML

3. any openapi version in json

```
Accept: application/openapi+yaml;version=3.1,
        application/openapi+yaml;version=3.0;q=0.5,
        application/openapi+json;q=0.3
```

2.3.1. Media Type application/openapi+json

The following information serves as the registration form for the application/openapi+json media type.

Type name: application

Subtype name: openapi+json

Required parameters: None

Optional parameters: version; unrecognized parameters should be ignored

Encoding considerations: Same as [JSON]

Security considerations: see Section 4 of this document

Interoperability considerations: None

Published specification: (this document)

Applications that use this media type: HTTP

Fragment identifier considerations: Same as for application/json [JSON]

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): json

Macintosh file type code(s): n/a

Person and email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: None.

Author: See Authors' Addresses section.

Change controller: n/a

2.3.2. Media Type application/openapi+yaml

The following information serves as the registration form for the application/openapi+yaml media type.

Type name: application

Subtype name: openapi+yaml

Required parameters: None

Optional parameters: version; unrecognized parameters should be ignored

Encoding considerations: Same as [JSON]

Security considerations: see Section 4 of this document

Interoperability considerations: see Section 2.1

Published specification: (this document)

Applications that use this media type: HTTP

Fragment identifier considerations: Same as for application/json [JSON]

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): yaml, yml

Macintosh file type code(s): n/a

Person and email address to contact for further information: See Authors' Addresses section

Intended usage: COMMON

Restrictions on usage: None.

Author: See Authors' Addresses section

Change controller: n/a

2.4. JSON Schema Media Types

JSON Schema is a declarative domain-specific language for validating and annotating JSON documents (see [jsonschema]).

This document registers the media types associated with JSON Schema.

There are many dialects of JSON Schema in wide use today. The JSON Schema maintainers have released several dialects including draft-04, draft-07, and draft 2020-12. There are also several third-party JSON Schema dialects in wide use including the ones defined for use in OpenAPI and MongoDB.

This specification defines little more than how to identify the dialect while leaving most of the semantics of the schema up to the dialect to define. Clients MUST use the following order of precedence for determining the dialect of a schema.

*The $schema keyword (Section 2.4.1)

*The "schema" media type parameter (Section 2.4.2)

*The context of the enclosing document. This applies only when a schema is embedded within a document. The enclosing document could be another schema in the case of a bundled schema or it could be another type of document that includes schemas such as an OpenAPI document.

*If none of the above result in identifying the dialect, client behavior is undefined.

2.4.1. The "$schema" Keyword

The $schema keyword is used as a JSON Schema dialect identifier. The value of this keyword MUST be a URI [RFC3986]. This URI SHOULD identify a meta-schema that can be used to validate that the schema is syntactically correct according to the dialect the URI identifies.

The dialect SHOULD define where the $schema keyword is allowed and/or recognized in a schema, but it is RECOMMENDED that dialects do not allow the schema to change within the same Schema Resource.

2.4.2. Identifying a Schema via a Media Type Parameter

Media types MAY allow for a schema media type parameter, to support content negotiation based on schema identifier (see Section 12 of [SEMANTICS]). The schema media type parameter MUST be a URI-reference [RFC3986].

The schema parameter identifies a schema that provides semantic information about the resource the media type represents. When using the application/schema+json media type, the schema parameter identifies the dialect of the schema the media type represents.

The schema URI is opaque and SHOULD NOT automatically be dereferenced. Since schema doesn't necessarily point to a network location, the "describedby" relation is used for linking to a downloadable schema.

The following is an example of content negotiation where a user agent can accept two different versions of a "pet" resource. Each resource version is identified by a unique JSON Schema.

Request:

```
NOTE: '\' line wrapping per RFC 8792

GET /pet/1234 HTTP/1.1
Host: foo.example
Accept: \
  application/schema-instance+json; schema="/schemas/v2/pet"; q=0.2, \
  application/schema-instance+json; schema="/schemas/v1/pet"; q=0.1
```

Response:

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 Ok
Content-Type: \
  application/schema-instance+json; schema="/schemas/v2/pet"

{
  "petId": "1234",
  "name": "Pluto",
  ...
}
```

In the following example, the user agent is able to accept two possible dialects of JSON Schema and the server replies with the latest one.

Request:

```
NOTE: '\' line wrapping per RFC 8792

GET /schemas/v2/pet HTTP/1.1
Host: foo.example
Accept: application/schema+json; \
            schema="https://json-schema.org/draft/2020-12/schema", \
        application/schema+json; \
            schema="http://json-schema.org/draft-07/schema#"
```

Response:

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: \
  application/schema+json; \
      schema="https://json-schema.org/draft/2020-12/schema"

{
  "$id": "https://json-schema.org/draft/2020-12/schema",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  ...
}
```

2.4.3. Linking to a Schema

It is RECOMMENDED that instances described by a schema provide a link to a downloadable JSON Schema using the link relation describedby, as defined by Linked Data Protocol 1.0, section 8.1 [W3C.REC-ldp-20150226].

In HTTP, such links can be attached to any response using the Link header [LINK].

```
Link: <https://example.com/my-hyper-schema#>; rel="describedby"
```

2.4.4. Fragment Identifiers

Two fragment identifier structures are supported: JSON Pointers and plain-names.

The use of JSON Pointers as URI fragment identifiers is described in [RFC6901]. Fragment identifiers that are empty or start with a /, MUST be interpreted as JSON Pointer fragment identifiers.

Plain-name fragment identifiers reference locally named locations in the document. The dialect determines how plain-name identifiers map to locations within the document. All fragment identifiers that do not match the JSON Pointer syntax MUST be interpreted as plain name fragment identifiers.

2.4.5. Media Type application/schema+json

The application/schema+json media type represents JSON Schema. This schema can be an official dialect or a third-party dialect. The following information serves as the registration form for the application/schema+json media type.

Type name: application

Subtype name: schema+json

Required parameters: N/A

Optional parameters:

- **schema**: A URI identifying the JSON Schema dialect the schema was written for. If this value conflicts with the value of the $schema keyword in the schema, the $schema keyword takes precedence.

Encoding considerations: Same as [JSON]

Security considerations: See the "Security Considerations" section of [jsonschema]

Interoperability considerations: See the "General Considerations" section of [jsonschema]

Published specification: (this document)

Applications that use this media type: JSON Schema is used in a variety of applications including API servers and clients that validate JSON requests and responses, IDEs that valid configuration files, databases that store JSON, and more.

Fragment identifier considerations: See Section 2.4.4

Additional information:

- **Deprecated alias names for this type**: N/A
- **Magic number(s)**: N/A
- **File extension(s)**: json, schema.json

  *Macintosh file type code(s): N/A

Person and email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: N/A.

Author: See Authors' Addresses section.

Change controller: N/A

2.4.6. Media Type application/schema-instance+json

The application/schema-instance+json media type is an extension of the [JSON] media type that just adds the schema media type parameter and fragment identification. The following information serves as the registration form for the application/schema-instance+json media type.

Type name: application

Subtype name: schema-instance+json

Required parameters: N/A

Optional parameters:

  *__schema__: A URI identifying a JSON Schema that provides semantic information about this JSON representation.

Encoding considerations: Same as [JSON]

Security considerations: Same as [JSON]

Interoperability considerations: Same as [JSON]

Published specification: (this document)

Applications that use this media type: JSON Schema is used in a variety of applications including API servers and clients that validate JSON requests and responses, IDEs that valid configuration files, databases that store JSON, and more.

Fragment identifier considerations: See Section 2.4.4

Additional information:

  *__Deprecated alias names for this type__: N/A

*   **Magic number(s)**: N/A

*   **File extension(s)**: json

*   **Macintosh file type code(s)**: N/A

Person and email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See Authors' Addresses section.

Change controller: N/A

3. Interoperability Considerations

3.1. YAML Media Types

3.1.1. YAML is an Evolving Language

YAML is an evolving language and, in time, some features have been added, and others removed.

While this document is based on a given YAML version [YAML], media types registration does not imply a specific version. This allows content negotiation of version-independent YAML resources.

Implementers concerned about features related to a specific YAML version can specify it in the documents using the %YAML directive (see Section 6.8.1 of [YAML]).

3.1.2. YAML and JSON

When using flow collection styles (see Section 7.4 of [YAML]) a YAML document could look like JSON [JSON], thus similar interoperability considerations apply.

When using YAML as a more efficient format to serialize information intented to be consumed as JSON, information can be discarded: this includes comments (see Section 3.2.3.3 of [YAML]) and alias nodes (see Section 7.1 of [YAML]), that do not have a JSON counterpart.

```
# This comment will be lost
# when serializing in JSON.
Title:
  type: string
  maxLength: &text_limit 64

Name:
  type: string
  maxLength: *text_limit  # Replaced by the value 64.
```

Figure 1: JSON replaces alias nodes with static values.

Implementers need to ensure that relevant information will not be lost during the processing. For example, they might consider acceptable that alias nodes are replaced by static values.

In some cases an implementer may want to define a list of allowed YAML features, taking into account that the following ones might have interoperability issues with JSON:

- non UTF-8 encoding, since YAML supports UTF-16 and UTF-32 in addition to UTF-8;
- mapping keys that are not strings;
- circular references represented using anchor (see Section 4.1.2 and Figure 3).
- .inf and .nan float values, since JSON does not support them;
- non-JSON types, including the ones associated to tags like !! timestamp that were deployed in older YAML versions;
- tags in general, and specifically ones that do not map to JSON types like custom and local tags such as !!python/object and ! mytag (see Section 2.4 of [YAML]);

```
non-json-keys:
  2020-01-01: a timestamp
  [0, 1]: a sequence
  ? {k: v}
  : a map
non-json-value: 2020-01-01
```

Figure 2: Example of mapping keys not supported in JSON

4. Security Considerations

Security requirements for both media type and media type suffix registrations are discussed in Section 4.6 of [MEDIATYPE].

4.1. YAML Media Types

4.1.1. Arbitrary Code Execution

Care should be used when using YAML tags, because their implementation might trigger unexpected code execution.

Code execution in deserializers should be disabled by default, and only be enabled explicitly. In those cases, the implementation should ensure - for example, via specific functions - that the code execution results in strictly bounded time/memory limits.

Many implementations provide safe deserializers addressing these issues.

4.1.2. Resource exhaustion

YAML documents are rooted, connected, directed graphs and can contain reference cycles, so they can't be treated as simple trees (see Section 3.2.1 of [YAML]). An implementation that attempts to do that can infinite-loop at some point (e.g. when trying to serialize a YAML document in JSON).

```
x: &x
  y: *x
```

Figure 3: A cyclic document

Even if a document is not cyclic, treating it as a tree could lead to improper behaviors (such as the "billion laughs" problem).

```
x1: &a1 ["a", "a"]
x2: &a2 [*a1, *a1]
x3: &a3 [*a2, *a2]
```

Figure 4: A billion laughs document

This can be addressed using processors limiting the anchor recursion depth and validating the input before processing it; even in these cases it is important to carefully test the implementation you are going to use. The same considerations apply when serializing a YAML representation graph in a format that do not support reference cycles (see Section 3.1.2).

5. IANA Considerations

This specification defines the following new Internet media types [MEDIATYPE].

IANA has updated the "Media Types" registry at https://www.iana.org/assignments/media-types with the registration information provided below.

Media Type	Section
application/yaml	Section 2.1 of ThisRFC
application/openapi+yaml	Section 2.3.2 of ThisRFC
application/openapi+json	Section 2.3.1 of ThisRFC
application/schema+json	Section 2.4.5 of ThisRFC
application/schema-instance+json	Section 2.4.6 of ThisRFC

Table 1

IANA has updated the "Structured Syntax Suffixes" registry at https://www.iana.org/assignments/media-type-structured-suffix with the registration information provided below.

Suffix	Section
+yaml	Section 2.2 of ThisRFC

Table 2

6. Normative References

[JSON] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[jsonschema] Wright, A., Andrews, H., Hutton, B., and G. Dennis, "JSON Schema Core", 28 January 2020, <https://json-schema.org/specification.html>.

[LINK] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

[MEDIATYPE] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[oas] Darrel Miller, Jeremy Whitlock, Marsh Gardiner, Mike Ralphson, Ron Ratovsky, and Uri Sarid, "OpenAPI Specification 3.0.0", 26 July 2017.

[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>.

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

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/rfc/rfc6901>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <https://www.rfc-editor.org/rfc/rfc7405>.

[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>.

[SEMANTICS] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>.

[W3C.REC-ldp-20150226] Speicher, S., Arwe, J., and A. Malhotra, "Linked Data Platform 1.0", World Wide Web Consortium Recommendation REC-ldp-20150226, 26 February 2015, <https://www.w3.org/TR/2015/REC-ldp-20150226>.

[YAML] Oren Ben-Kiki, Clark Evans, and Ingy dot Net, "YAML Ain't Markup Language Version 1.2", 1 October 2021, <https://yaml.org/spec/1.2/spec.html>.

Appendix A. Acknowledgements

Thanks to Erik Wilde and David Biesack for being the initial contributors of this specification, and to Darrel Miller and Rich Salz for their support during the adoption phase.

In addition to the people above, this document owes a lot to the extensive discussion inside and outside the HTTPAPI workgroup. The following contributors have helped improve this specification by opening pull requests, reporting bugs, asking smart questions, drafting or reviewing text, and evaluating open issues:

Eemeli Aro, Tina (tinita) Mueller, Ben Hutton and Jason Desrosiers.

FAQ

Q: Why this document? After all these years, we still lack a proper media-type for YAML. This has some security implications too (eg. wrt on identifying parsers or treat downloads)

Change Log

RFC EDITOR PLEASE DELETE THIS SECTION.

Author's Address

Roberto Polli
Digital Transformation Department, Italian Government
Italy

Email: robipolli@gmail.com