Workgroup: HTTPAPI
Internet-Draft:
draft-ietf-httpapi-yaml-mediatypes-03
Published: 5 August 2022
Intended Status: Informational
Expires: 6 February 2023
Authors:
R. Polli
Digital Transformation Department, Italian Government
E. Wilde
Axway
E. Aro
Mozilla

YAML Media Type

Abstract

This document registers the application/yaml media type and the +yaml structured syntax suffix on the IANA Media Types registry.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-httpapi-yaml-mediatypes/.

Discussion of this document takes place on the HTTPAPI Working Group mailing list (mailto:httpapi@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/httpapi/. Subscribe at https://www.ietf.org/mailman/listinfo/httpapi/. Working Group information can be found at https://datatracker.ietf.org/wg/httpapi/about/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-httpapi/mediatypes/labels/yaml.

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 6 February 2023.

Table of Contents

1. Introduction

YAML [YAML] is a data serialization format that is capable of conveying one or multiple documents in a single presentation stream (e.g. a file or a network resource). It is widely used on the Internet, including in the API sector (e.g. see [OAS]), but the relevant media type and structured syntax suffix previously had not been registered by IANA.

To increase interoperability when exchanging YAML streams, and leverage content negotiation mechanisms when exchanging YAML resources, this specification registers the application/yaml media type and the +yaml structured syntax suffix.

Moreover, it provides security considerations and interoperability considerations related to [YAML], including its relation with [JSON].

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].

The terms "fragment" and "fragment identifier" in this document are to be interpreted as in [URI].

The terms "presentation", "stream", "YAML document", "representation graph", "tag", "node", "alias node", "anchor" and "anchor name" in this document are to be interpreted as in [YAML].

1.2. Fragment identification

A fragment identifies a node in a stream.

A fragment identifier starting with "*" is to be interpreted as a YAML alias node Section 1.2.1.

For single-document YAML streams, a fragment identifier that is empty or that starts with "/" is to be interpreted as a JSON Pointer [JSON-POINTER] and is evaluated on the YAML representation graph, walking through alias nodes; in particular, the empty fragment identifier references the root node. This syntax can only reference the YAML nodes that are on a path that is made up of nodes interoperable with the JSON data model (see Section 3.3).

A fragment identifier is not guaranteed to reference an existing node. Therefore, applications SHOULD define how an unresolved alias node ought to be handled.

1.2.1. Fragment identification via alias nodes

This section describes how to use alias nodes (see Section 3.2.2.2 and 7.1 of [YAML]) as fragment identifiers to designate nodes.

A YAML alias node can be represented in a URI fragment identifier by encoding it into bytes using UTF-8 [UTF-8], while percent-encoding those characters not allowed by the fragment rule in Section 3.5 of [URI].

If multiple nodes would match a fragment identifier, the first such match is selected.

Users concerned with interoperability of fragment identifiers:

- SHOULD limit alias nodes to a set of characters that do not require encoding to be expressed as URI fragment identifiers: this is generally possible since anchor names are a serialization detail;

- SHOULD NOT use alias nodes that match multiple nodes.

In the example resource below, the URL file.yaml#*foo references the first alias node *foo pointing to the node with value scalar and not the one in the second document; whereas the URL file.yaml#*document_2 references the root node of the second document { one: [a, sequence]}.

```
%YAML 1.2
---
one: &foo scalar
two: &bar
  - some
  - sequence
  - items
...
%YAML 1.2
---
&document_2
one: &foo [a, sequence]
```

Figure 1: A YAML stream containing two YAML documents.

2. Media Type and Structured Syntax Suffix registrations

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

2.1. Media Type application/yaml

The media type for YAML text is application/yaml; the following information serves as the registration form for this media type.

Type name: application

Subtype name: yaml

Required parameters: N/A

Optional parameters: N/A; unrecognized parameters should be ignored

Encoding considerations: binary

Security considerations: see Section 4 of this document

Interoperability considerations: see Section 3 of this document

Published specification: [YAML]

Applications that use this media type: Applications that need a human-friendly, cross language, Unicode based data serialization language designed around the common native data types of dynamic programming languages.

Fragment identifier considerations: See Section 1.2

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: IESG

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 Language (YAML)

+suffix: +yaml

References: [YAML]

Encoding considerations: see Section 2.1

Fragment identifier considerations: Differently from application/yaml, there is no fragment identification syntax defined for +yaml.

A specific xxx/yyy+yaml media type needs to define the syntax and semantics for fragment identifiers because the ones in Section 2.1 do not apply unless explicitly expressed.

Interoperability considerations: See Section 2.1

Security considerations: See Section 2.1

Contact: httpapi@ietf.org or art@ietf.org

Author:
See Authors' Addresses section

Change controller: IESG

3. Interoperability Considerations

3.1. YAML is an Evolving Language

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

While this document is based on a given YAML version [YAML], the media type 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 YAML documents using the %YAML directive (see Section 6.8.1 of [YAML]).

3.2. YAML streams

A YAML stream can contain zero or more YAML documents.

When receiving a multi-document stream, an application that only expects one-document streams, ought to signal an error instead of ignoring the extra documents.

Current implementations consider different documents in a stream independent, similarly to JSON Text Sequences (see [RFC7464]); elements such as anchors are not guaranteed to be referenceable across different documents.

3.3. 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 intended to be consumed as JSON, information not reflected in the representation graph and classified as presentation or serialization detail (see Section 3.2 of [YAML]) can be discarded. This includes comments (see Section 3.2.3.3 of [YAML]), directives, 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 2: 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:

- multi-document YAML streams;
- 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 4.2 and Figure 4);
- .inf and .nan float values, since JSON does not support them;
- non-JSON types, including the ones associated with tags like !! timestamp that were included in the default schema of older YAML versions;
- tags in general, and specifically the 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]);

```
%YAML 1.2
---
non-json-keys:
  0: a number
  [0, 1]: a sequence
  ? {k: v}
  : a map
---
non-json-keys:
  !date 2020-01-01: a timestamp
non-json-value: !date 2020-01-01
...
```

Figure 3: Example of mapping keys and values not supported in JSON in a multi-document YAML stream

3.4. Fragment identifiers

To allow fragment identifiers to traverse alias nodes, the YAML representation graph needs to be generated before the fragment identifier evaluation. It is important that this evaluation will not cause the issues mentioned in Section 3.3 and in Security considerations (Section 4) such as infinite loops and unexpected code execution.

Implementers need to consider that the YAML version and supported features (e.g. merge keys) can impact on the generation of the representation graph (see Figure 9).

In Section 2.1, this document extends the use of specifications based on the JSON data model with support for YAML fragment identifiers. This is to improve the interoperability of already consolidated practices, such as the one of writing OpenAPI documents [OAS] in YAML.

Appendix A provides a non-exhaustive list of examples that could help understand interoperability issues related to fragment identifiers.

4. Security Considerations

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

4.1. Arbitrary Code Execution

Care should be used when using YAML tags, because their resolution 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.

4.2. 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 traversing the YAML representation graph at some point, for example:

- when trying to serialize it JSON;
- or when searching/identifying nodes using specifications based on the JSON data model (e.g. [JSON-POINTER]).

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

Figure 4: A cyclic document

Even if a representaion graph is not cyclic, treating it as a simple 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 5: 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 does not support reference cycles (see Section 3.3).

4.3. YAML streams

Incremental parsing and processing of a YAML stream can produce partial results and later indicate failure to parse the remainder of

the stream; to prevent partial processing, implementers might prefer validating all the documents in a stream beforehand.

Repeated parsing and re-encoding of a YAML stream can result in the addition or removal of document delimiters (e.g. --- or ...) as well as the modification of anchor names and other serialization details: this can break signature validation.

5. IANA Considerations

This specification defines the following new Internet media type [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 this document

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 this document

Table 2

6. References

6.1. 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>.

[JSON-POINTER] 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>.

[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>.

[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>.

[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>.

[URI] 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>.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/rfc/rfc3629>.

[YAML] Oren Ben-Kiki, Clark Evans, Ingy dot Net, Tina Müller, Pantelis Antoniou, Eemeli Aro, and Thomas Smith, "YAML Ain't Markup Language Version 1.2", 1 October 2021, <https://yaml.org/spec/1.2.2/>.

6.2. Informative References

[I-D.ietf-jsonpath-base] Gössner, S., Normington, G., and C. Bormann, "JSONPath: Query expressions for JSON", Work in Progress, Internet-Draft, draft-ietf-jsonpath-base-05, 25

April 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-jsonpath-base-05>.

[RFC7464] Williams, N., "JavaScript Object Notation (JSON) Text Sequences", RFC 7464, DOI 10.17487/RFC7464, February 2015, <https://www.rfc-editor.org/rfc/rfc7464>.

Appendix A. Examples related to fragment identifier interoperability

A.1. Unreferenceable nodes

In this example, a couple of YAML nodes that cannot be referenced based on the JSON data model since their mapping keys are not strings.

```
%YAML 1.2
---
a-map-cannot:
  ? {be: expressed}
  : with a JSON Pointer

0: no numeric mapping keys in JSON
```

Figure 6: Example of YAML nodes that are not referenceable based on JSON data model.

A.2. Referencing a missing node

In this example the fragment #/0 does not reference an existing node

```
0: "JSON Pointer `#/0` references a string mapping key."
```

Figure 7: Example of a JSON Pointer that does not reference an existing node.

A.3. Representation graph with anchors and cyclic references

In this YAML document, the #/foo/bar/baz fragment identifier traverses the representation graph and references the string you. Moreover, the presence of a cyclic reference implies that there are infinite fragment identifiers #/foo/bat/../bat/bar referencing the &anchor node.

```
anchor: &anchor
  baz: you
foo: &foo
  bar: *anchor
  bat: *foo
```

Figure 8: Example of a cyclic references and alias nodes.

Many YAML implementations will resolve the merge key "<<:" defined in YAML 1.1 in the representation graph. This means that the fragment #/book/author/given_name references the string Federico and that the fragment #/book/<< will not reference any existing node.

```
%YAML 1.1
---
# Many implementations use merge keys.
the-viceroys: &the-viceroys
  title: The Viceroys
  author:
    given_name: Federico
    family_name: De Roberto
book:
  <<: *the-viceroys
  title: The Illusion
```

Figure 9: Example of YAML merge keys.

Appendix B. 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:

Tina (tinita) Mueller, Ben Hutton, Manu Sporny and Jason Desrosiers.

FAQ

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

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)

Q: Why using alias nodes as fragment identifiers?

Alias nodes are a native YAML feature that allows addressing any node in a YAML document. Since YAML is not limited to string keywords, not all nodes are addressable using JSON Pointers. Alias nodes are thus the natural choice for fragment identifiers Section 1.2.

Q: Why not use plain names for alias nodes? Why not define plain names?

Using plain name fragments could have limited the ability of future xxx+yaml media types to define their plain name fragments. Moreover, alias nodes starts with * so we found no reason to strip it when using them in fragments.

Preserving * had another positive result: it allows distinguishing a fragment identifier expressed as an alias node from one expressed in other formats. In this document we included JSON Pointer [JSON-POINTER] which is expected to start with /. Moreover, since JSON Path [I-D.ietf-jsonpath-base] expressions start with $, this mechanism can be extended to JSON Path too.

Q: Why not just use JSON Pointer as the primary fragment identifier?

Fragment identifiers in YAML always reference YAML representation graph nodes. JSON Pointer can only rely on string keywords so it is not able to reference a generic node in the representation graph.

Since JSON Pointer is a specification unrelated to YAML, we decided to isolate the impacts of changes in JSON Pointer on YAML fragments: only fragments starting with "/" are "delegated" to an external spec, and if [JSON-POINTER] changes, it will only affect fragments starting with "/".

The current behaviour for empty fragments is the same for both JSON Pointer and alias nodes. Incidentally, it's the only sensible behaviour independently of [JSON-POINTER].

Change Log

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

Since draft-ietf-httpapi-yaml-mediatypes-02

* clarification on fragment identifiers #50.

Since draft-ietf-httpapi-yaml-mediatypes-01

* application/yaml fragment identifiers compatible with JSON Pointer #41 (#47).

Authors' Addresses

Roberto Polli
Digital Transformation Department, Italian Government
Italy

Email: robipolli@gmail.com

Erik Wilde
Axway
Switzerland

Email: erik.wilde@dret.net

Eemeli Aro
Mozilla
Finland

Email: eemeli@gmail.com