YAML Media Type
RFC 9512

Status   Email expansions   History

Document		
	Type	RFC - Informational (February 2024)
		Was draft-ietf-httpapi-yaml-mediatypes (httpapi WG)
	Authors	Roberto Polli ✉, Erik Wilde ✉, Eemeli Aro ✉
	Last updated	2024-02-14
	RFC stream	Internet Engineering Task Force (IETF)
	Formats	txt html xml pdf htmlized bibtex
	Additional resources	Mailing list discussion
IESG	**Responsible AD**	Zaheduzzaman Sarker ✉
	Send notices to	(None)

✉ Email authors   ✉ Email WG   ⚡ IPR   ← References   → Referenced by   Search Lists

---

RFC 9512

```
Internet Engineering Task Force (IETF)                          R. Polli
Request for Comments: 9512                      DTD, Italian Government
Category: Informational                                         E. Wilde
ISSN: 2070-1721                                                    Axway
                                                                  E. Aro
                                                                Mozilla
                                                          February 2024


                          YAML Media Type


Abstract


   This document registers the application/yaml media type and the +yaml
   structured syntax suffix with IANA.  Both identify document
```

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 a
   corresponding media type and structured syntax suffix had not
   previously 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 [MEDIATYPE].

   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.

   The terms "content negotiation" and "resource" in this document are
   to be interpreted as in [HTTP].

   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", "serialization detail", "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 (see 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,
traversing 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 3.4).

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 Sections 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], but percent-encoding of
those characters is not allowed by the fragment rule in Section 3.5
of [URI].

If multiple nodes match a fragment identifier, the first occurrence
of such a 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), and

*   SHOULD NOT use alias nodes that match multiple nodes.

In the example resource below, the relative reference (see
Section 4.2 of [URI]) file.yaml#*foo identifies the first alias node
*foo pointing to the node with value scalar and not to the one in the
second document, whereas the relative reference file.yaml#*document_2
identifies the root node of the second document {one: [a, sequence]}.

```
 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 includes the information required for IANA to register
   the application/yaml media type and the +yaml structured syntax
   suffix per [MEDIATYPE].

2.1. Media Type application/yaml

   The media type for YAML 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], this document

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

   Fragment identifier considerations: See Section 1.2 of this
      document.

       Deprecated alias names for this type: application/x-yaml, text/
          yaml, and text/x-yaml. These names are used but are not
          registered.
       Magic number(s): N/A
       File extension(s): "yaml" (preferred) and "yml". See Section 3.3
          of this document.
       Macintosh file type code(s): N/A
       Windows Clipboard Name: YAML

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

    Intended usage: COMMON

    Restrictions on usage: None

    Author: See the Authors' Addresses section of this document.

    Change controller: IETF

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 structured syntax
    suffix registration form follows. See [MEDIATYPE] for definitions of
    each part of the registration form.

    Name: YAML Ain't Markup Language (YAML)

    +suffix: +yaml

    References: [YAML], this document

    Encoding considerations: Same as application/yaml

    Interoperability considerations: Same as application/yaml

    Fragment identifier considerations: Unlike 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 defined for
       application/yaml do not apply unless explicitly expressed.

    Security considerations: Same as application/yaml

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

Change controller: IETF

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.

   The application/yaml media type registration is independent of the
   YAML 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 single-document streams should 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. Filename Extension

   The "yaml" filename extension is the preferred one; it is the most
   popular and widely used on the web. The "yml" filename extension is
   still used. The simultaneous usage of two filename extensions in the
   same context might cause interoperability issues (e.g., when both a
   "config.yaml" and a "config.yml" are present).

3.4. 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

```
%YAML 1.2
---
# 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 processing. For example, they might consider alias nodes
being replaced by static values as acceptable.

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

* multi-document YAML streams

* non-UTF-8 encoding. Before encoding YAML streams in UTF-16 or
   UTF-32, it is important to note that Section 8.1 of [JSON]
   mandates the use of UTF-8 when exchanging JSON texts between
   systems that are not part of a closed ecosystem and that
   Section 5.2 of [YAML] recommends the use of UTF-8.

* mapping keys that are not strings

* cyclic references represented using anchors (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, specifically ones that do not map to JSON types,
   e.g., custom and local tags such as !!python/object and !mytag
   (see Section 2.4 of [YAML])

```
%YAML 1.2
```

```
          [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

3.5. 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 does not
cause the issues mentioned in Sections 3.4 and 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 affect the generation of the
representation graph (see Figure 9).

In Section 1.2, 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 writing OpenAPI documents [OAS] in YAML.

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

4. Security Considerations

Security requirements for both media types and media type suffixes
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 the latter case, the implementation
should ensure (for example, via specific functions) that the code
execution results in strictly bounded time/memory limits.

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 treats them as
   simple trees risks going into an infinite loop while traversing the
   YAML representation graph. This can happen:

   * when trying to serialize it as JSON or

   * when searching/identifying nodes using specifications based on the
      JSON data model (e.g., [JSON-POINTER]).

```
%YAML 1.2
---
x: &x
   y: *x
```

                        Figure 4: A Cyclic Document

   Even if a representation graph is not cyclic, treating it as a simple
   tree could lead to improper behaviors, such as triggering an
   Exponential Data Expansion (e.g., a Billion Laughs Attack).

```
%YAML 1.2
---
x1: &a1 ["a", "a"]
x2: &a2 [*a1, *a1]
x3: &a3 [*a2, *a2]
```

                     Figure 5: A Billion Laughs Document

   This can be addressed using processors that limit the anchor
   recursion depth and validate 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 3.4).

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 and processing all the documents in a stream at the same
   time.

that can break signature validation.

4.4. Expressing Booleans

Section 10.3.2 of [YAML] specifies that only the scalars matching the regular expression true|True|TRUE|false|False|FALSE are interpreted as booleans. Older YAML versions were more tolerant (e.g., interpreting NO and N as False and interpreting YES and Y as True). When the older syntax is used, a YAML implementation could then interpret {insecure: n} as {insecure: "n"} instead of {insecure: false}. Using the syntax defined in Section 10.3.2 of [YAML] prevents these issues.

5. IANA Considerations

IANA has updated the "Media Types" registry (https://www.iana.org/assignments/media-types) with the registration information in Section 2.1 for the media type application/yaml.

IANA has updated the "Structured Syntax Suffixes" registry (https://www.iana.org/assignments/media-type-structured-suffix) with the registration information in Section 2.2 for the structured syntax suffix +yaml.

6. References

6.1. Normative References

[HTTP]   Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke,
            Ed., "HTTP Semantics", STD 97, RFC 9110,
            DOI 10.17487/RFC9110, June 2022,
            <https://www.rfc-editor.org/info/rfc9110>.

[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/info/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/info/rfc6901>.

[MEDIATYPE]
            Freed, N., Klensin, J., and T. Hansen, "Media Type
            Specifications and Registration Procedures", BCP 13,

[OAS]   Miller, D., Whitlock, J., Gardiner, M., Ralphson, M.,
           Ratovsky, R., and U. Sarid, "OpenAPI Specification",
           v3.0.0, 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/info/rfc2119>.

[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/info/rfc8174>.

[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/info/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/info/rfc3629>.

[YAML]   Ben-Kiki, O., Evans, C., dot Net, I., Müller, T.,
           Antoniou, P., Aro, E., and T. Smith, "YAML Ain't Markup
           Language Version 1.2", 1 October 2021,
           <https://yaml.org/spec/1.2.2/>.

6.2. Informative References

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

Appendix A. Examples Related to Fragment Identifier Interoperability

A.1. Unreferenceable Nodes

This example shows 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
```

on JSON Data Model

A.2. Referencing a Missing Node

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

```
%YAML 1.2
---
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.

```
%YAML 1.2
---
anchor: &anchor
  baz: you
foo: &foo
  bar: *anchor
  bat: *foo
```

          Figure 8: Example of a Cyclic Reference and Alias Nodes

Many YAML implementations will resolve the merge key "<<:"
(https://yaml.org/type/merge.html) 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:
```

                  Figure 9: Example of YAML Merge Keys

Acknowledgements

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

    In addition, this document owes a lot to the extensive discussion
    inside and outside the HTTPAPI Working Group. The following
    contributors helped improve this specification by opening pull
    requests, reporting bugs, asking smart questions, drafting or
    reviewing text, and evaluating open issues: Tina (tinita) Müller, Ben
    Hutton, Carsten Bormann, Manu Sporny, and Jason Desrosiers.

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