A JSON Encoding for HTTP Header Field Values
draft-reschke-http-jfv-09

Abstract

   This document establishes a convention for use of JSON-encoded field
   values in HTTP header fields.

Editorial Note (To be removed by RFC Editor before publication)

   Distribution of this document is unlimited. Although this is not a
   work item of the HTTPbis Working Group, comments should be sent to
   the Hypertext Transfer Protocol (HTTP) mailing list at ietf-http-
   wg@w3.org [1], which may be joined by sending a message with subject
   "subscribe" to ietf-http-wg-request@w3.org [2].

   Discussions of the HTTPbis Working Group are archived at
   <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

   XML versions and latest edits for this document are available from
   <http://greenbytes.de/tech/webdav/#draft-reschke-http-jfv>.

   The changes in this draft are summarized in Appendix E.12.

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 January 3, 2019.

Copyright Notice

Table of Contents

1. Introduction

Defining syntax for new HTTP header fields ([RFC7230], Section 3.2)
is non-trivial. Among the commonly encountered problems are:

o There is no common syntax for complex field values. Several well-
   known header fields do use a similarly looking syntax, but it is
   hard to write generic parsing code that will both correctly handle
   valid field values but also reject invalid ones.

o The HTTP message format allows header fields to repeat, so field
   syntax needs to be designed in a way that these cases are either
   meaningful, or can be unambiguously detected and rejected.

o HTTP/1.1 does not define a character encoding scheme ([RFC6365],
   Section 2), so header fields are either stuck with US-ASCII
   ([RFC0020]), or need out-of-band information to decide what
   encoding scheme is used. Furthermore, APIs usually assume a
   default encoding scheme in order to map from octet sequences to
   strings (for instance, [XMLHttpRequest] uses the IDL type
   "ByteString", effectively resulting in the ISO-8859-1 character
   encoding scheme [ISO-8859-1] being used).

(See Section 8.3.1 of [RFC7231] for a summary of considerations for
new header fields.)

This specification addresses the issues listed above by defining both
a generic JSON-based ([RFC8259]) data model and a concrete wire
format that can be used in definitions of new header fields, where
the goals were:

o to be compatible with header field recombination when fields occur
   multiple times in a single message (Section 3.2.2 of [RFC7230]),
   and

o   not to use any problematic characters in the field value (non-
    ASCII characters and certain whitespace characters).

    Note: [HSTRUCT], a work item of the IETF HTTP Working Group, is a
    different attempt to address this set of problems -- it tries to
    identify and formalize common field structures in existing header
    fields; the syntax defined over there would usually lead to a more
    compact notation.

2. Data Model and Format

In HTTP, header fields with the same field name can occur multiple
times within a single message (Section 3.2.2 of [RFC7230]). When
this happens, recipients are allowed to combine the field values
using commas as delimiter. This rule matches nicely JSON's array
format (Section 5 of [RFC8259]). Thus, the basic data model used
here is the JSON array.

Header field definitions that need only a single value can restrict
themselves to arrays of length 1, and are encouraged to define error
handling in case more values are received (such as "first wins",
"last wins", or "abort with fatal error message").

JSON arrays are mapped to field values by creating a sequence of
serialized member elements, separated by commas and optionally
whitespace. This is equivalent to using the full JSON array format,
while leaving out the "begin-array" ('[') and "end-array" (']')
delimiters.

The ABNF character names and classes below are used (copied from
[RFC5234], Appendix B.1):

```
CR                  = %x0D    ; carriage return
HTAB                = %x09    ; horizontal tab
LF                  = %x0A    ; line feed
SP                  = %x20    ; space
VCHAR               = %x21-7E ; visible (printing) characters
```

Characters in JSON strings that are not allowed or discouraged in
HTTP header field values -- that is, not in the "VCHAR" definition --
need to be represented using JSON's "backslash" escaping mechanism
([RFC8259], Section 7).

The control characters CR, LF, and HTAB do not appear inside JSON
strings, but can be used outside (line breaks, indentation etc.).
These characters need to be either stripped or replaced by space
characters (ABNF "SP").

Formally, using the HTTP specification's ABNF extensions defined in
Section 7 of [RFC7230]:

```
json-field-value = #json-field-item
json-field-item  = JSON-Text
                   ; see [RFC8259], Section 2,
                   ; post-processed so that only VCHAR characters
                   ; are used
```

3. Sender Requirements

To map a JSON array to an HTTP header field value, process each array
element separately by:

1. generating the JSON representation,

2. stripping all JSON control characters (CR, HTAB, LF), or
    replacing them by space ("SP") characters,

3. replacing all remaining non-VSPACE characters by the equivalent
    backslash-escape sequence ([RFC8259], Section 7).

The resulting list of strings is transformed into an HTTP field value
by combining them using comma (%x2C) plus optional SP as delimiter,
and encoding the resulting string into an octet sequence using the
US-ASCII character encoding scheme ([RFC0020]).

4. Recipient Requirements

To map a set of HTTP header field instances to a JSON array:

1. combine all header field instances into a single field as per
    Section 3.2.2 of [RFC7230],

2. add a leading begin-array ("[") octet and a trailing end-array
    ("]") octet, then

3. run the resulting octet sequence through a JSON parser.

The result of the parsing operation is either an error (in which case
the header field values needs to be considered invalid), or a JSON
array.

5. Using this Format in Header Field Definitions

Specifications defining new HTTP header fields need to take the
considerations listed in Section 8.3.1 of [RFC7231] into account.

Many of these will already be accounted for by using the format
defined in this specification.

Readers of HTTP-related specifications frequently expect an ABNF
definition of the field value syntax. This is not really needed
here, as the actual syntax is JSON text, as defined in Section 2 of
[RFC8259].

A very simple way to use this JSON encoding thus is just to cite this
specification -- specifically the "json-field-value" ABNF production
defined in Section 2 -- and otherwise not to talk about the details
of the field syntax at all.

An alternative approach is just to repeat the ABNF-related parts from
Section 2.

This frees the specification from defining the concrete on-the-wire
syntax. What's left is defining the field value in terms of a JSON
array. An important aspect is the question of extensibility, e.g.
how recipients ought to treat unknown field names. In general, a
"must ignore" approach will allow protocols to evolve without
versioning or even using entire new field names.

6. Deployment Considerations

This JSON-based syntax will only apply to newly introduced header
fields, thus backwards compatibility is not a problem. That being
said, it is conceivable that there is existing code that might trip
over double quotes not being used for HTTP's quoted-string syntax
(Section 3.2.6 of [RFC7230]).

7. Interoperability Considerations

The "I-JSON Message Format" specification ([RFC7493]) addresses known
JSON interoperability pain points. This specification borrows from
the requirements made over there:

7.1. Encoding and Characters

This specification requires that field values use only US-ASCII
characters, and thus by definition use a subset of UTF-8 (Section 2.1
of [RFC7493]).

7.2. Numbers

Be aware of the issues around number precision, as discussed in
Section 2.2 of [RFC7493].

7.3. Object Constraints

   As described in Section 4 of [RFC8259], JSON parser implementations
   differ in the handling of duplicate object names. Therefore, senders
   MUST NOT use duplicate object names, and recipients SHOULD either
   treat field values with duplicate names as invalid (consistent with
   [RFC7493], Section 2.3) or use the lexically last value (consistent
   with [ECMA-262], Section 24.3.1.1).

   Furthermore, ordering of object members is not significant and can
   not be relied upon.

8. Internationalization Considerations

   In HTTP/1.1, header field values are represented by octet sequences,
   usually used to transmit ASCII characters, with restrictions on the
   use of certain control characters, and no associated default
   character encoding, nor a way to describe it ([RFC7230],
   Section 3.2). HTTP/2 does not change this.

   This specification maps all characters which can cause problems to
   JSON escape sequences, thereby solving the HTTP header field
   internationalization problem.

   Future specifications of HTTP might change to allow non-ASCII
   characters natively. In that case, header fields using the syntax
   defined by this specification would have a simple migration path (by
   just stopping to require escaping of non-ASCII characters).

9. Security Considerations

   Using JSON-shaped field values is believed to not introduce any new
   threads beyond those described in Section 12 of [RFC8259], namely the
   risk of recipients using the wrong tools to parse them.

   Other than that, any syntax that makes extensions easy can be used to
   smuggle information through field values; however, this concern is
   shared with other widely used formats, such as those using parameters
   in the form of name/value pairs.

10. References

10.1. Normative References

   [RFC0020] Cerf, V., "ASCII format for network interchange", STD 80,
              RFC 20, DOI 10.17487/RFC0020, October 1969,
              <https://www.rfc-editor.org/info/rfc20>.

[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/info/rfc5234>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
           Protocol (HTTP/1.1): Message Syntax and Routing",
           RFC 7230, DOI 10.17487/RFC7230, June 2014,
           <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
           Protocol (HTTP/1.1): Semantics and Content", RFC 7231,
           DOI 10.17487/RFC7231, June 2014,
           <https://www.rfc-editor.org/info/rfc7231>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493,
           DOI 10.17487/RFC7493, March 2015,
           <https://www.rfc-editor.org/info/rfc7493>.

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

10.2. Informative References

[CLEARSITE]
           West, M., "Clear Site Data", W3C Working Draft WD-clear-
           site-data-20171130, November 2017,
           <https://www.w3.org/TR/2017/WD-clear-site-data-20171130/>.

           Latest version available at <https://www.w3.org/TR/clear-
           site-data/>.

[ECMA-262]
           Ecma International, "ECMA-262 6th Edition, The ECMAScript
           2015 Language Specification", Standard ECMA-262, June
           2015, <http://www.ecma-international.org/ecma-262/6.0/>.

[FEATUREPOL]
           Clelland, I., "Feature Policy", W3C Draft Community Group
           Report , June 2018,
           <https://wicg.github.io/feature-policy/>.

[HSTRUCT] Nottingham, M. and P-H. Kamp, "Structured Headers for
           HTTP", draft-ietf-httpbis-header-structure-07 (work in
           progress), July 2018.

   [ISO-8859-1]
              International Organization for Standardization,
              "Information technology -- 8-bit single-byte coded graphic
              character sets -- Part 1: Latin alphabet No. 1", ISO/
              IEC 8859-1:1998, 1998.

   [KEY]   Fielding, R. and M. Nottingham, "The Key HTTP Response
              Header Field", draft-ietf-httpbis-key-01 (work in
              progress), March 2016.

   [REPORTING]
              Grigorik, I. and M. West, "Reporting API 1", W3C Group
              Note NOTE-reporting-1-20160607, June 2016,
              <http://www.w3.org/TR/2016/NOTE-reporting-1-20160607/>.

              Latest version available at <http://www.w3.org/TR/
              reporting-1/>.

   [RFC6266] Reschke, J., "Use of the Content-Disposition Header Field
              in the Hypertext Transfer Protocol (HTTP)", RFC 6266,
              DOI 10.17487/RFC6266, June 2011,
              <https://www.rfc-editor.org/info/rfc6266>.

   [RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in
              Internationalization in the IETF", BCP 166, RFC 6365,
              DOI 10.17487/RFC6365, September 2011,
              <https://www.rfc-editor.org/info/rfc6365>.

   [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
              Protocol (HTTP/1.1): Authentication", RFC 7235,
              DOI 10.17487/RFC7235, June 2014,
              <https://www.rfc-editor.org/info/rfc7235>.

   [RFC8187] Reschke, J., "Indicating Character Encoding and Language
              for HTTP Header Field Parameters", RFC 8187,
              DOI 10.17487/RFC8187, September 2017,
              <https://www.rfc-editor.org/info/rfc8187>.

   [XMLHttpRequest]
              WhatWG, "XMLHttpRequest", <https://xhr.spec.whatwg.org/>.

10.3. URIs

   [1] mailto:ietf-http-wg@w3.org

   [2] mailto:ietf-http-wg-request@w3.org?subject=subscribe

Appendix A. Examples

   This section shows how some of the existing HTTP header fields would
   look like if they would use the format defined by this specification.

A.1. Content-Length

   "Content-Length" is defined in Section 3.3.2 of [RFC7230], with the
   field value's ABNF being:

      Content-Length = 1*DIGIT

   So the field value is similar to a JSON number ([RFC8259],
   Section 6).

   Content-Length is restricted to a single field instance, as it
   doesn't use the list production (as per Section 3.2.2 of [RFC7230]).
   However, in practice multiple instances do occur, and the definition
   of the header field does indeed discuss how to handle these cases.

   If Content-Length was defined using the JSON format discussed here,
   the ABNF would be something like:

      Content-Length = #number
                     ; number: [RFC8259], Section 6

   ...and the prose definition would:

   o restrict all numbers to be non-negative integers without
      fractions, and

   o require that the array of values is of length 1 (but allow the
      case where the array is longer, but all members represent the same
      value)

A.2. Content-Disposition

   Content-Disposition field values, defined in [RFC6266], consist of a
   "disposition type" (a string), plus multiple parameters, of which at
   least one ("filename") sometime needs to carry non-ASCII characters.

   For instance, the first example in Section 5 of [RFC6266]:

      Attachment; filename=example.html

   has a disposition type of "Attachment", with filename parameter value
   "example.html". A JSON representation of this information might be:

```
   {
     "Attachment": {
       "filename" : "example.html"
     }
   }
```

which would translate to a header field value of:

```
   { "Attachment": { "filename" : "example.html" } }
```

The third example in Section 5 of [RFC6266] uses a filename parameter
containing non-US-ASCII characters:

```
   attachment; filename*=UTF-8''%e2%82%ac%20rates
```

Note that in this case, the "filename*" parameter uses the encoding
defined in [RFC8187], representing a filename starting with the
Unicode character U+20AC (EURO SIGN), followed by " rates". If the
definition of Content-Disposition would have used the format proposed
here, the workaround involving the "parameter*" syntax would not have
been needed at all.

The JSON representation of this value could then be:

```
   { "attachment": { "filename" : "\u20AC rates" } }
```

A.3. WWW-Authenticate

The WWW-Authenticate header field value is defined in Section 4.1 of
[RFC7235] as a list of "challenges":

```
    WWW-Authenticate = 1#challenge
```

...where a challenge consists of a scheme with optional parameters:

```
   challenge   = auth-scheme [ 1*SP ( token68 / #auth-param ) ]
```

An example for a complex header field value given in the definition
of the header field is:

```
   Newauth realm="apps", type=1, title="Login to \"apps\"",
   Basic realm="simple"
```

(line break added for readability)

A possible JSON representation of this field value would be the array
below:

```
   [
     {
       "Newauth" : {
         "realm": "apps",
         "type" : 1,
         "title" : "Login to \"apps\""
       }
     },
     {
       "Basic" : {
         "realm": "simple"
       }
     }
   ]
```

...which would translate to a header field value of:

```
   { "Newauth" : { "realm": "apps", "type" : 1,
                   "title": "Login to \"apps\"" }},
   { "Basic" : { "realm": "simple"}}
```

A.4. Accept-Encoding

The Accept-Encoding header field value is defined in Section 5.3.4 of [RFC7231] as a list of codings, each of which allowing a weight parameter 'q':

```
    Accept-Encoding = #( codings [ weight ] )
    codings         = content-coding / "identity" / "*"
    weight          = OWS ";" OWS "q=" qvalue
    qvalue          = ( "0" [ "." 0*3DIGIT ] )
                    / ( "1" [ "." 0*3("0") ] )
```

An example for a complex header field value given in the definition of the header field is:

```
   gzip;q=1.0, identity; q=0.5, *;q=0
```

Due to the defaulting rules for the quality value ([RFC7231], Section 5.3.1), this could also be written as:

```
   gzip, identity; q=0.5, *; q=0
```

A JSON representation could be:

```
  [
    {
      "gzip" : {
      }
    },
    {
      "identity" : {
         "q": 0.5
      }
    },
    {
      "*" : {
         "q": 0
      }
    }
  ]
```

...which would translate to a header field value of:

```
  {"gzip": {}}, {"identity": {"q": 0.5}}, {"*": {"q": 0}}
```

In this example, the part about "gzip" appears unnecessarily verbose,
as the value is just an empty object. A simpler notation would
collapse members like these to string literals:

```
  "gzip", {"identity": {"q": 0.5}}, {"*": {"q": 0}}
```

If this is desirable, the header field definition could allow both
string literals and objects, and define that a mere string literal
would be mapped to a member whose name is given by the string
literal, and the value is an empty object.

For what it's worth, one of the most common cases for 'Accept-
Encoding' would become:

```
  "gzip", "deflate"
```

which would be only a small overhead over the original format.

Appendix B. Use of JSON Field Value Encoding in the Wild

Since work started on this document, various specifications have
adopted this format. At least one of these moved away after the HTTP
Working Group decided to focus on [HSTRUCT] (see thread starting at
<https://lists.w3.org/Archives/Public/ietf-http-
wg/2016OctDec/0505.html>).

The sections below summarize the current usage of this format.

B.1. W3C Reporting API Specification

Defined in W3C Note "Reporting API 1" (Section 3.1 of [REPORTING]). Still in use in latest editor copy as of June 2017.

B.2. W3C Clear Site Data Specification

Used in earlier versions of "Clear Site Data". The current version replaces the use of JSON with a custom syntax that happens to be somewhat compatible with an array of JSON strings (see Section 3.1 of [CLEARSITE] and <https://lists.w3.org/Archives/Public/ietf-http-wg/2017AprJun/0214.html> for feedback).

B.3. W3C Feature Policy Specification

Originally defined in W3C Draft Community Group Report "Feature Policy" ([FEATUREPOL]), but now replaced with a custom syntax (see <https://github.com/WICG/feature-policy/pull/83>).

Appendix C. Relation to HTTP 'Key' Header Field

[KEY] aims to improve the cacheability of responses that vary based on certain request header fields, addressing lack of granularity in the existing "Vary" response header field ([RFC7231], Section 7.1.4). If the JSON-based format described by this document gains popularity, it might be useful to add a JSON-aware "Key Parameter" (see Section 2.3 of [KEY]).

Appendix D. Discussion

This approach uses a default of "JSON array", using implicit array markers. An alternative would be a default of "JSON object". This would simplify the syntax for non-list-typed header fields, but all the benefits of having the same data model for both types of header fields would be gone. A hybrid approach might make sense, as long as it doesn't require any heuristics on the recipient's side.

   Note: a concrete proposal was made by Kazuho Oku in <https://lists.w3.org/Archives/Public/ietf-http-wg/2016JanMar/0155.html>.

[[CREF1: Use of generic libs vs compactness of field values..]]

Appendix E. Change Log (to be removed by RFC Editor before publication)

E.1. **Since draft-reschke-http-jfv-00**

   Editorial fixes + working on the TODOs.

E.2. **Since draft-reschke-http-jfv-01**

   Mention slightly increased risk of smuggling information in header
   field values.

E.3. **Since draft-reschke-http-jfv-02**

   Mention Kazuho Oku's proposal for abbreviated forms.

   Added a bit of text about the motivation for a concrete JSON subset
   (ack Cory Benfield).

   Expand I18N section.

E.4. **Since draft-reschke-http-jfv-03**

   Mention relation to KEY header field.

E.5. **Since draft-reschke-http-jfv-04**

   Between June and December 2016, this was a work item of the HTTP
   working group (see <https://datatracker.ietf.org/doc/draft-ietf-
   httpbis-jfv/>). Work (if any) continues now on
   <https://datatracker.ietf.org/doc/draft-reschke-http-jfv/>.

   Changes made while this was a work item of the HTTP Working Group:

E.6. **Since draft-ietf-httpbis-jfv-00**

   Added example for "Accept-Encoding" (inspired by Kazuho's feedback),
   showing a potential way to optimize the format when default values
   apply.

E.7. **Since draft-ietf-httpbis-jfv-01**

   Add interop discussion, building on I-JSON and ECMA-262 (see
   <https://github.com/httpwg/http-extensions/issues/225>).

E.8. **Since draft-ietf-httpbis-jfv-02**

   Move non-essential parts into appendix.

   Updated XHR reference.

E.9. Since draft-reschke-http-jfv-05

Add meat to "Using this Format in Header Field Definitions".

Add a few lines on the relation to "Key".

Summarize current use of the format.

E.10. Since draft-reschke-http-jfv-06

RFC 5987 is obsoleted by RFC 8187.

Update CLEARSITE comment.

E.11. Since draft-reschke-http-jfv-07

Update JSON and HSTRUCT references.

FEATUREPOL doesn't use JSON syntax anymore.

E.12. Since draft-reschke-http-jfv-08

Update HSTRUCT reference.

Update notes about CLEARSITE and FEATUREPOL.

Acknowledgements

Thanks go to the Hypertext Transfer Protocol Working Group participants.

Author's Address

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/