HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 27, 2017

J. Reschke
greenbytes

October 24, 2016

A JSON Encoding for HTTP Header Field Values
draft-ietf-httpbis-jfv-02

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)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/>.

Working Group information can be found at <http://httpwg.github.io/>; source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions>.

The changes in this draft are summarized in Appendix A.

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 http://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 April 27, 2017.

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:

- 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.

- 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.

- 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 ([RFC7159]) data model and a concrete wire format that can be used in definitions of new header fields, where the goals were:

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

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

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 [RFC7159]). 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 ([RFC7159], 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 [RFC7159], 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 ([RFC7159], 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

[[anchor5: Explain what a definition of a new header field needs to do precisely to use this format, mention must-ignore extensibility]]

6. 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.

6.1. 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 ([RFC7159], 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: [RFC7159], Section 6
```

...and the prose definition would:

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

- 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)

6.2. 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 [RFC5987], 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" } }
```

6.3. 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"}}
```

6.4. 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.

7. 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>.

[[anchor7: Use of generic libs vs compactness of field values..]]

[[anchor8: Mention potential "Key" header field extension ([KEY]).]]

8. 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]).

9. Interoperability Considerations

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

9.1. 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]).

9.2. Numbers

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

9.3. Object Constraints

As described in Section 4 of [RFC7159], 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.

10. 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).

11. Security Considerations

Using JSON-shaped field values is believed to not introduce any new threads beyond those described in Section 12 of [RFC7159], 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.

12. References

12.1. Normative References

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <http://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, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc7231>.

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

12.2. Informative References

[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/>.

[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.

[RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, DOI 10.17487/RFC5987, August 2010, <http://www.rfc-editor.org/info/rfc5987>.

[RFC6266] Reschke, J., "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)", RFC 6266, DOI 10.17487/RFC6266, June 2011, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7235>.

[XMLHttpRequest] van Kesteren, A., Aubourg, J., Song, J., and H. Steen, "XMLHttpRequest Level 1", W3C Working Draft WD-XMLHttpRequest-20140130, January 2014, <http://www.w3.org/TR/2014/WD-XMLHttpRequest-20140130/>.

Latest version available at <http://www.w3.org/TR/XMLHttpRequest/>.

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

A.1. Since draft-reschke-http-jfv-00

Editorial fixes + working on the TODOs.

A.2. Since draft-reschke-http-jfv-01

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

A.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.

A.4. Since draft-reschke-http-jfv-03

Mention relation to KEY header field.

A.5. Since draft-reschke-http-jfv-04

Change to HTTP Working Group draft.

A.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.

A.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>).

Appendix B. 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/