Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2015

J. Reschke
greenbytes

July 2, 2014

A JSON Encoding for HTTP Header Field Values
draft-reschke-http-jfv-00

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>.

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 January 3, 2015.

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:

- 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 fail on 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 ([USASCII]), 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.

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 lenght 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 can 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

[[anchor3: The text below assumes we're starting with a JSON-formatted sequence of characters, not octets; need to clarify.]] 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.

4. Recipient Requirements

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

1. remove all header field instances that only contain whitespace (SP / HTAB) and "comma" characters [[anchor5: either drop this or make it more precise]],

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

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

4. 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

[[anchor7: Explain what a definition of a new header field needs to do precisely to use this format]]

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 ([RFC7230], 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 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"}}
```

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 haeders, 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.

[[anchor9: Use of generic libs vs compactness of field values..]]

8. Deployment Considerations

[[anchor11: Mention that some code might be refused by double quotes not being used for quoted-string.]]

9. Internationalization Considerations

[[anchor13: TBD, mention migration path to message format that is robust wrt UTF-8, or other binary encodings of JSON]]

10. Security Considerations

[[anchor15: TBD]]

11. References

11.1. Normative References

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

11.2. Informative References

[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.

[RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, August 2010.

[RFC6266] Reschke, J., "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)", RFC 6266, June 2011.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

[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/>.

URIs

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

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

Appendix A. Sample Code and Test Cases

[[anchor19: TBD]]

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/