Workgroup: Network Working Group
Published: 22 April 2021
Intended Status: Informational
Expires: 24 October 2021
A J. F. Reschke
ugreenbytes
t
h
o
r
s
:

A JSON Encoding for HTTP Field Values

Abstract

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

Editorial Note

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

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, which may be joined by sending a message with subject "subscribe" to ietf-http-wg-request@w3.org.

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 D.17.

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 24 October 2021.

Copyright Notice

Table of Contents

1. Introduction

Defining syntax for new HTTP fields ([HTTP], Section 5) is non-trivial. Among the commonly encountered problems are:

- There is no common syntax for complex field values. Several well-known 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 field lines 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 does not define a character encoding scheme ([RFC6365], Section 2), so 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 16.3 of [HTTP] for a summary of considerations for new 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 fields, where the goals were:

- to be compatible with field recombination when field lines occur multiple times in a single message (Section 5.3 of [HTTP]), and

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

1.1. Relation to "Structured Field Values for HTTP" (*[RFC8941]*)

"Structured Field Values for HTTP", an IETF RFC on the Standards Track, is a different approach to this set of problems. It uses a more compact notation, similar to what is used in existing header fields, and avoids several potential interoperability problems inherent to the use of JSON.

In general, that format is preferred for newly defined fields. The JSON-based format defined by this document might however be useful in case the data that needs to be transferred is already in JSON format, or features not covered by "Structured Field Values" are needed.

See Appendix A for more details.

2. Data Model and Format

In HTTP, field lines with the same field name can occur multiple times within a single message (Section 5.3 of [HTTP]). When this happens, recipients are allowed to combine the field line values using commas as delimiter, forming a combined "field value". This rule matches nicely JSON's array format (Section 5 of [RFC8259]). Thus, the basic data model used here is the JSON array.

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 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 5.6.1 of [HTTP]:

```
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 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]).

3.1. Example

With the JSON data below, containing the non-ASCII characters "ü" (LATIN SMALL LETTER U WITH DIAERESIS, U+00FC) and "€" (EURO SIGN, U+20AC):

```
[
  {
    "destination": "Münster",
    "price": 123,
    "currency": "€"
  }
]
```

The generated field value would be:

```
{ "destination": "M\u00FCnster", "price": 123, "currency": "\u20AC" }
```

4. Recipient Requirements

To map a set of HTTP field line values to a JSON array:

1. combine all field line values into a single field value as per Section 5.3 of [HTTP],

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 field values needs to be considered invalid), or a JSON array.

4.1. Example

An HTTP message containing the field lines:

```
Example: "\u221E"
Example: {"date":"2012-08-25"}
Example: [17,42]
```

would be parsed into the JSON array below:

```
[
  "∞",
  {
    "date": "2012-08-25"
  },
  [
    17,
    42
  ]
]
```

5. Using this Format in Field Definitions

Specifications defining new HTTP fields need to take the considerations listed in Section 16.3 of [HTTP] 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 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 5.6.4 of [HTTP]).

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 uses a subset of UTF-8 (Section 2.1 of [RFC7493]).

Furthermore, escape sequences in JSON strings (Section 7 of [RFC8259]) - both in object member names and string values - are not allowed to represent non-Unicode code points such as unpaired surrogates or Noncharacters (see "General Structure" in [UNICODE]).

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 are not allowed to use duplicate object names, and recipients are advised to 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 current versions of HTTP, 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 ([HTTP], Section 5).

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

Future specifications of HTTP might change to allow non-ASCII characters natively. In that case, 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

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-15, 30 March 2021, <https://tools.ietf.org/html/draft-ietf-httpbis-semantics-15>.

[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>.

[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>.

[UNICODE] The Unicode Consortium, "The Unicode Standard", <http://www.unicode.org/versions/latest/>.

10.2. 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.

[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>.

[RFC8941] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/info/rfc8941>.

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

10.3. Specifications Using This Syntax (at some point of time)

[CLEARSITE]
West, M., "Clear Site Data", W3C Working Draft WD-clear-site-data-20171130, 30 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/>.

[FEATUREPOL] Clelland, I., "Feature Policy", W3C Editor's Draft , <https://w3c.github.io/webappsec-feature-policy/>.

[REPORTING] Creager, D., Grigorik, I., Meyer, P., and M. West, "Reporting API", W3C Working Draft WD-reporting-1-20180925, 25 September 2018, <https://www.w3.org/TR/2018/WD-reporting-1-20180925/>. Latest version available at <https://www.w3.org/TR/reporting-1/>.

Appendix A. Comparison with Structured Fields

A.1. Base Types

Type	in Structured Fields	in JSON-based Fields
Integer	[RFC8941], Section 3.3.1	[RFC8259], Section 6
	(restricted to 15 digits)	
Decimal	[RFC8941], Section 3.3.2	[RFC8259], Section 6
	(a fixed point decimal restricted to 12 + 3 digits)	
String	[RFC8941], Section 3.3.3	[RFC8259], Section 7
	(only ASCII supported, non-ASCII requires using Byte Sequences)	
Token	[RFC8941], Section 3.3.4	not available
Byte Sequence	[RFC8941], Section 3.3.5	not available
		(usually mapped to strings using base64 encoding)
Boolean	[RFC8941], Section 3.3.6	[RFC8259], Section 3

Table 1

Structured Fields provide more data types (such as "token" or "byte sequence"). Numbers are restricted, avoiding the JSON interop problems described in Section 7.2. Strings are limited to ASCII, requiring the use of byte sequences should non-ASCII characters be needed.

A.2. Structures

Structured Fields define Lists ([RFC8941], Section 3.1), similar to JSON arrays ([RFC8259], Section 5), and Dictionaries ([RFC8941], Section 3.2), similar to JSON objects ([RFC8259], Section 4).

In addition, most items in Structured Fields can be parametrized ([RFC8941], Section 3.1.2), attaching a dictionary-like structure to the value. To emulate this in JSON based field, an additional nesting of objects would be needed.

Finally, nesting of data structures is intentionally limited to two levels (see Appendix A.1 of [RFC8941] for the motivation).

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

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

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 [RFC8941] (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 Working Draft "Reporting API" (Section 3.1 of [REPORTING]). Still in use in latest working draft dated September 2018.

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 document "Feature Policy" ([FEATUREPOL]), but switched to use of Structured Header Fields ([RFC8941]).

Appendix C. Implementations

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

See <https://github.com/reschke/json-fields> for a proof-of-concept (in development).

Appendix D. Change Log

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

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

Editorial fixes + working on the TODOs.

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

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

D.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.

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

Mention relation to KEY header field.

D.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:

D.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.

D.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>).

D.8. Since draft-ietf-httpbis-jfv-02

Move non-essential parts into appendix.

Updated XHR reference.

D.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.

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

RFC 5987 is obsoleted by RFC 8187.

Update CLEARSITE comment.

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

Update JSON and HSTRUCT references.

FEATUREPOL doesn't use JSON syntax anymore.

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

Update HSTRUCT reference.

Update notes about CLEARSITE and FEATUREPOL.

D.13. Since draft-reschke-http-jfv-09

Update HSTRUCT and FEATUREPOL references.

Update note about REPORTING.

Changed category to "informational".

D.14. Since draft-reschke-http-jfv-10

Update HSTRUCT reference.

D.15. Since draft-reschke-http-jfv-11

Update HSTRUCT reference.

Update note about FEATUREPOL (now using Structured Fields).

Reference [HTTP] instead if RFC723* and adjust (header) field terminology accordingly.

Remove discussion about the relation to KEY (as that spec is dormant: <https://datatracker.ietf.org/doc/draft-ietf-httpbis-key/>).

Remove appendices "Examples" and "Discussion".

Mark "Use of JSON Field Value Encoding in the Wild" for removal in RFC.

D.16. Since draft-reschke-http-jfv-12

Update HTTP reference and update terminology some more.

Update HSTRUCT reference (now RFC 8941).

D.17. Since draft-reschke-http-jfv-13

Update HTTP reference.

Mention test implementation.

Clarify that Unicode unpaired surrogates or Noncharacters must not be sent.

Rewrite text about [RFC8941], add appendix comparing both formats.

And send/receive examples.

Acknowledgements

Thanks go to the Hypertext Transfer Protocol Working Group participants.

Author's Address

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
48155 Münster
Germany

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