Network Working Group   M. Nottingham
Internet-Draft   Fastly
Intended status: Standards Track   November 1, 2019
Expires: May 4, 2020

Binary Structured HTTP Headers
draft-nottingham-binary-structured-headers-00

Abstract

This specification defines a binary serialisation of Structured Headers for HTTP, along with a negotiation mechanism for its use in HTTP/2. It also defines how to use Structured Headers for many existing headers - thereby "backporting" them - when supported by two peers.

Note to Readers

RFC EDITOR: please remove this section before publication

The issues list for this draft can be found at https://github.com/mnot/I-D/labels/binary-structured-headers [1].

The most recent (often, unpublished) draft is at https://mnot.github.io/I-D/binary-structured-headers/ [2].

Recent changes are listed at https://github.com/mnot/I-D/commits/gh-pages/binary-structured-headers [3].

See also the draft's current status in the IETF datatracker, at https://datatracker.ietf.org/doc/draft-nottingham-binary-structured-headers/ [4].

Status of This Memo

Table of Contents

1. Introduction

HTTP messages often pass through several systems - clients, intermediaries, servers, and subsystems of each - that parse and process their header and trailer fields. This repeated parsing (and often re-serialisation) adds latency and consumes CPU, energy, and other resources.

Structured Headers for HTTP [I-D.ietf-httpbis-header-structure] offers a set of data types that new headers can combine to express their semantics. This specification defines a binary serialisation of those structures in Section 2, and specifies its use in HTTP/2 - specifically, as part of HPACK Literal Header Field Representations ([RFC7541]) - in Section 3.

Section 4 defines how to use Structured Headers for many existing headers when supported by two peers.

The primary goal of this specification are to reduce parsing overhead and associated costs, as compared to the textual representation of Structured Headers. A secondary goal is a more compact wire format in common situations. An additional goal is to enable future work on more granular header compression mechanisms.

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.

2. Binary Structured Headers

This section defines a binary serialisation for the Structured Header Types defined in [I-D.ietf-httpbis-header-structure].

The types permissable as the top-level of Structured Header field values - Dictionary, List, and Item - are defined in terms of a Binary Literal Representation (Section 2.1), which is a replacement for the String Literal Representation in [RFC7541].

Binary representations of the remaining types are defined in Section 2.2.

2.1. The Binary Literal Representation

The Binary Literal Representation is a replacement for the String Literal Representation defined in [RFC7541], Section 5.2, for use in BINHEADERS frames (Section 3.2).

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|   Type (4)    | PLength (4+)  |
+---+---------------------------+
| Payload Data (Length octets)  |
+-------------------------------+
```

A binary literal representation contains the following fields:

- Type: Four bits indicating the type of the payload.

- PLength: The number of octets used to represent the payload, encoded as per [RFC7541], Section 5.1, with a 4-bit prefix.

- Payload Data: The payload, as per below.

The following payload types are defined:

2.1.1. Lists

List values (type=0x1) have a payload consisting of a stream of Binary Structured Types representing the members of the list. Members that are Items are represented as per Section 2.2.3; members that are inner-lists are represented as per Section 2.2.1.

If any member cannot be represented, the entire field value MUST be serialised as a String Literal (Section 2.1.4).

2.1.2. Dictionaries

Dictionary values (type=0x2) have a payload consisting of a stream of members.

Each member is represented by a key length, followed by that many bytes of the member-name, followed by Binary Structured Types representing the member-value.

```
  0   1   2   3   4   5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
| KL (8+)                       |  member-name (KL octets)
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---
| member-value
+---+---+---+---+---+---+---+---
```

A parameter's fields are:

- KL: The number of octets used to represent the member-name, encoded as per [RFC7541], Section 5.1, with a 8-bit prefix

- member-name: KL octets of the member-name

- member-value: One or more Binary Structure Types

member-values that are Items are represented as per Section 2.2.3; member-values that are inner-lists are represented as per Section 2.2.1.

If any member cannot be represented, the entire field value MUST be serialised as a String Literal (Section 2.1.4).

2.1.3. Items

Item values (type=0x3) have a payload consisting of Binary Structured Types, as described in Section 2.2.3.

2.1.4. String Literals

String Literals (type=0x4) are the string value of a header field; they are used to carry header field values that are not Binary Structured Headers, and may not be Structured Headers at all. As such, their semantics are that of String Literal Representations in [RFC7541], Section 5.2.

Their payload is the octets of the field value.

ISSUE: use Huffman coding? https://github.com/mnot/I-D/issues/305 [5]

2.2. Binary Structured Types

Every Binary Structured Type starts with a 5-bit type field that identifies the format of its payload:

```
  0   1   2   3   4   5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
      Type (5)      |  Payload...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
```

Some Binary Structured Types contain padding bits; senders MUST set padding bits to 0; recipients MUST ignore their values.

2.2.1. Inner Lists

The Inner List data type (type=0x1) has a payload in the format:

```
  5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---
     L(3+)  |  Members (L octets)
+---+---+---+---+---+---+---+---+---+---+---
```

Its fields are:

o L: The number of octets used to represent the members, encoded as per [RFC7541], Section 5.1, with a 3-bit prefix

o Members: L octets

Each member of the list will be represented as an Item (Section 2.2.3); if any member cannot, the entire field value will be serialised as a String Literal (Section 2.1.4).

The inner list's parameters, if present, are serialised in a following Parameter type (Section 2.2.2); they do not form part of the payload of the inner list.

2.2.2. Parameters

The Parameters data type (type=0x2) has a payload in the format:

```
  5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---
     L(3+)  |  Parameters (L octets)
+---+---+---+---+---+---+---+---+---+---+---
```

Its fields are:

o L: The number of octets used to represent the token, encoded as per [RFC7541], Section 5.1, with a 3-bit prefix

o Parameters: L octets

Each parameter is represented by key length, followed by that many bytes of the parameter-name, followed by a Binary Structured Type representing the parameter-value.

```
  0   1   2   3   4   5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
| KL (8+)                       |  parameter-name (KL octets)
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---
| parameter-value (VL octets)
+---+---+---+---+---+---+---+---
```

A parameter's fields are:

- KL: The number of octets used to represent the parameter-name, encoded as per [RFC7541], Section 5.1, with a 8-bit prefix

- parameter-name: KL octets of the parameter-name

- parameter-value: A Binary Structured type representing a bare item (Section 2.2.3)

Parameter-values are bare items; that is, they MUST NOT have parameters themselves.

If the parameters cannot be represented, the entire field value will be serialised as a String Literal (Section 2.1.4).

Parameters are always associated with the Binary Structured Type that immediately preceded them. If parameters are not explicitly allowed on the preceding type, or there is no preceding type, it is an error.

ISSUE: use Huffman coding for parameter-name? https://github.com/mnot/I-D/issues/305 [6]

2.2.3. Item Payload Types

Individual Structured Header Items can be represented using the Binary Payload Types defined below.

The item's parameters, if present, are serialised in a following Parameter type (Section 2.2.2); they do not form part of the payload of the item.

2.2.3.1. Integers

The Integer data type (type=0x3) has a payload in the format:

```
  5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---
  S |   X   | Length (8+)
+---+---+---+---+---+---+---+---+---+---+---

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---
|  Integer (Length octets)
+---+---+---+---+---+---+---+---
```

Its fields are:

- S: sign bit; 0 is negative, 1 is positive
- X: 2 bits of padding
- Length: The number of octets used to represent the integer, encoded as per [RFC7541], Section 5.1, with a 2-bit prefix
- Integer: Length octets

2.2.3.2. Floats

The Float data type (type=0x4) have a payload in the format:

```
  5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---
  S |   X   | ILength (8+)
+---+---+---+---+---+---+---+---+---+---+---

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---
|  Integer (ILength octets)
+---+---+---+---+---+---+---+---

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---
|  FLength (8+)
+---+---+---+---+---+---+---+---

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---
|  Fractional (FLength octets)
+---+---+---+---+---+---+---+---
```

Its fields are:

- S: sign bit; 0 is negative, 1 is positive
- X: 2 bits of padding
- ILength: The number of octets used to represent the integer component, encoded as per [RFC7541], Section 5.1, with a 2-bit prefix.
- Integer - ILength octets
- FLength: The number of octets used to represent the fractional component, encoded as per [RFC7541], Section 5.1, with a 2-bit prefix.
- Fractional: FLength octets

2.2.3.3. Strings

The String data type (type=0x5) has a payload in the format:

```
  5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---
     L(3+)  |  String (L octets)
+---+---+---+---+---+---+---+---+---+---+---
```

Its fields are:

- L: The number of octets used to represent the string, encoded as per [RFC7541], Section 5.1, with a 3-bit prefix.
- String: L octets.

ISSUE: use Huffman coding? https://github.com/mnot/I-D/issues/305 [7]

2.2.3.4. Tokens

The Token data type (type=0x6) has a payload in the format:

```
  5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---
     L(3+)  |  Token (L octets)
+---+---+---+---+---+---+---+---+---+---+---
```

Its fields are:

- L: The number of octets used to represent the token, encoded as per [RFC7541], Section 5.1, with a 3-bit prefix.

- Token: L octets.

ISSUE: use Huffman coding? https://github.com/mnot/I-D/issues/305 [8]

2.2.3.5. Byte Sequences

The Byte Sequence data type (type=0x7) has a payload in the format:

```
  5   6   7   0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+---+---+---
     L(3+)  |  Byte Sequence (L octets)
+---+---+---+---+---+---+---+---+---+---+---
```

Its fields are:

- L: The number of octets used to represent the byte sequence, encoded as per [RFC7541], Section 5.1, with a 3-bit prefix.

- Byte Sequence: L octets.

2.2.3.6. Booleans

The Boolean data type (type=0x8) has a payload of two bits:

```
  5   6   7
+---+---+---+
  B |   X   |
+---+---+---+
```

If B is 0, the value is False; if B is 1, the value is True. X is padding.

3. Using Binary Structured Headers in HTTP/2

When both peers on a connection support this specification, they can take advantage of that knowledge to serialise headers that they know to be Structured Headers (or compatible with them; see Section 4).

Peers advertise and discover this support using a HTTP/2 setting defined in Section 3.1, and convey Binary Structured Headers in a frame type defined in Section 3.2.

3.1. Binary Structured Headers Setting

Advertising support for Binary Structured Headers is accomplished using a HTTP/2 setting, SETTINGS_BINARY_STRUCTURED_HEADERS (0xTODO).

Receiving SETTINGS_BINARY_STRUCTURED_HEADERS from a peer indicates that:

1. The peer supports the Binary Structured Types defined in Section 2.

2. The peer will process the BINHEADERS frames as defined in Section 3.2.

3. When a downstream consumer does not likewise support that encoding, the peer will transform them into HEADERS frames (if the peer is HTTP/2) or a form it will understand (e.g., the textual representation of Structured Headers data types defined in [I-D.ietf-httpbis-header-structure]).

4. The peer will likewise transform all fields defined as Aliased Fields (Section 4.2) into their non-aliased forms as necessary.

The default value of SETTINGS_BINARY_STRUCTURED_HEADERS is 0. Future extensions to Structured Headers might use it to indicate support for new types.

3.2. The BINHEADERS Frame

When a peer has indicated that it supports this specification {#setting}, a sender can send the BINHEADERS Frame Type (0xTODO).

The BINHEADERS Frame Type behaves and is represented exactly as a HEADERS Frame type ([RFC7540], Section 6.2), with one exception; instead of using the String Literal Representation defined in [RFC7541], Section 5.2, it uses the Binary Literal Representation defined in Section 2.1.

Fields that are Structured Headers can have their values represented using the Binary Literal Representation corresponding to that header's top-level type - List, Dictionary, or Item; their values will then be serialised as a stream of Binary Structured Types.

Additionally, any field (including those defined as Structured Headers) can be serialised as a String Literal (Section 2.1.4), which accommodates headers that are not defined as Structured Headers, not valid Structured Headers, or that the sending implementation does not wish to send as Binary Structured Types for some other reason.

Note that Field Names are always serialised as String Literals (Section 2.1.4).

This means that a BINHEADERS frame can be converted to a HEADERS frame by converting the field values to the string representations of the various Structured Headers Types, and String Literals (Section 2.1.4) to their string counterparts.

Conversely, a HEADERS frame can be converted to a BINHEADERS frame by encoding all of the Literal field values as Binary Structured Types. In this case, the header types used are informed by the implementations knowledge of the individual header field semantics; see Section 4. Those which it cannot (do to either lack of knowledge or an error) or does not wish to convert into Structured Headers are conveyed in BINHEADERS as String Literals (Section 2.1.4).

Field values are stored in the HPACK [RFC7541] dynamic table without Huffman encoding, although specific Binary Structured Types might specify the use of such encodings.

Note that BINHEADERS and HEADERS frames MAY be mixed on the same connection, depending on the requirements of the sender. Also, note that only the field values are encoded as Binary Structured Types; field names are encoded as they are in HPACK.

4. Using Binary Structured Headers with Existing Fields

Any header field can potentially be parsed as a Structured Header according to the algorithms in [I-D.ietf-httpbis-header-structure] and serialised as a Binary Structured Header. However, many cannot, so optimistically parsing them can be expensive.

This section identifies fields that will usually succeed in Section 4.1, and those that can be mapped into Structured Headers by using an alias field name in Section 4.2.

4.1. Directly Represented Fields

The following HTTP field names can have their values parsed as Structured Headers according to the algorithms in [I-D.ietf-httpbis-header-structure], and thus can usually be serialised using the corresponding Binary Structured Types.

When one of these fields' values cannot be represented using Structured Types, its value can instead be represented as a String Literal (Section 2.1.4).

o Accept - List

- Accept-Encoding - List
- Accept-Language - List
- Accept-Patch - List
- Accept-Ranges - List
- Access-Control-Allow-Credentials - Item
- Access-Control-Allow-Headers - List
- Access-Control-Allow-Methods - List
- Access-Control-Allow-Origin - Item
- Access-Control-Max-Age - Item
- Access-Control-Request-Headers - List
- Access-Control-Request-Method - Item
- Age - Item
- Allow - List
- ALPN - List
- Alt-Svc - List
- Alt-Used - Item
- Cache-Control - Dictionary
- Content-Encoding - Item
- Content-Language - List
- Content-Length - Item
- Content-Type - Item
- Expect - Item
- Forwarded - List
- Host - Item

- Origin - Item
- Pragma - Dictionary
- Prefer - Dictionary
- Preference-Applied - Dictionary
- Retry-After - Item (see caveat below)
- Surrogate-Control - Dictionary
- TE - List
- Trailer - List
- Transfer-Encoding - List
- Vary - List
- X-Content-Type-Options - Item

Note that only the delta-seconds form of Retry-After is supported; a Retry-After value containing a http-date will need to be either converted into delta-seconds or serialised as a String Literal (Section 2.1.4).

4.2. Aliased Fields

The following HTTP field names can have their values represented in Structured headers by mapping them into its data types and then serialising the resulting Structured Header using an alternative field name.

For example, the Date HTTP header field carries a http-date, which is a string representing a date:

```
Date: Sun, 06 Nov 1994 08:49:37 GMT
```

Its value is more efficiently represented as an integer number of delta seconds from the Unix epoch (00:00:00 UTC on 1 January 1970, minus leap seconds). Thus, the example above would be represented in (non-binary) Structured headers as:

```
SH-Date: 784072177
```

As with directly represented fields, if the intended value of an aliased field cannot be represented using Structured Types

successfully, its value can instead be represented as a String Literal (Section 2.1.4).

Note that senders MUST know that the next-hop recipient understands these fields (typically, using the negotiation mechanism defined in Section 3) before using them. Likewise, recipients MUST transform them back to their unaliased form before forwarding the message to a peer or other consuming components that do not have this capability.

Each field name listed below indicates a replacement field name and a way to map its value to Structured Headers.

ISSUE: using separate names assures that the different syntax doesn't "leak" into normal headers, but it isn't strictly necessary if implementations always convert back to the correct form when giving it to peers or consuming software that doesn't understand this. https://github.com/mnot/I-D/issues/307 [9]

4.2.1. URLs

The following field names (paired with their replacement field names) have values that can be represented in Binary Structured Headers by considering their payload a string.

o Content-Location - SH-Content-Location

o Location - SH-Location

o Referer - SH-Referer

For example, a (non-binary) Location:

SH-Location: "https://example.com/foo"

TOOD: list of strings, one for each path segment, to allow better compression in the future?

4.2.2. Dates

The following field names (paired with their replacement field names) have values that can be represented in Binary Structured Headers by parsing their payload according to [RFC7230], Section 7.1.1.1, and representing the result as an integer number of seconds delta from the Unix Epoch (00:00:00 UTC on 1 January 1970, minus leap seconds).

o Date - SH-Date

o Expires - SH-Expires

- If-Modified-Since - SH-IMS
- If-Unmodified-Since - SH-IUS
- Last-Modified - SH-LM

For example, a (non-binary) Expires:

```
SH-Expires: 1571965240
```

4.2.3. ETags

The following field names (paired with their replacement field names) have values that can be represented in Binary Structured Headers by representing the entity-tag as a string, and the weakness flag as a boolean "w" parameter on it, where true indicates that the entity-tag is weak; if 0 or unset, the entity-tag is strong.

- ETag - SH-ETag

For example, a (non-Binary) ETag:

```
SH-ETag: "abcdef"; w=?1
```

If-None-Match is a list of the structure described above.

- If-None-Match - SH-INM

For example, a (non-binary) If-None-Match:

```
SH-INM: "abcdef"; w=?1, "ghijkl"
```

4.2.4. Links

The field-value of the Link header field [RFC8288] can be represented in Binary Structured Headers by representing the URI-Reference as a string, and link-param as parameters.

- Link: SH-Link

For example, a (non-binary) Link:

```
SH-Link: "/terms"; rel="copyright"; anchor="#foo"
```

4.2.5. Cookies

The field-value of the Cookie and Set-Cookie fields [RFC6265] can be represented in Binary Structured Headers as a List with parameters and a Dictionary, respectively. The serialisation is almost identical, except that the Expires parameter is always a string (as it can contain a comma), multiple cookie-strings can appear in Set-Cookie, and cookie-pairs are delimited in Cookie by a comma, rather than a semicolon.

Set-Cookie: SH-Set-Cookie Cookie: SH-Cookie

```
SH-Set-Cookie: lang=en-US, Expires="Wed, 09 Jun 2021 10:18:14 GMT"
SH-Cookie: SID=31d4d96e407aad42, lang=en-US
```

ISSUE: explicitly convert Expires to an integer? https://github.com/mnot/I-D/issues/308 [10]

5. IANA Considerations

ISSUE: todo

6. Security Considerations

As is so often the case, having alternative representations of data brings the potential for security weaknesses, when attackers exploit the differences between those representations and their handling.

One mitigation to this risk is the strictness of parsing for both non-binary and binary Structured Headers data types, along with the "escape valve" of String Literals (Section 2.1.4). Therefore, implementation divergence from this strictness can have security impact.

7. References

7.1. Normative References

[I-D.ietf-httpbis-header-structure]
Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-14 (work in progress), October 2019.

[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>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[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>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/info/rfc7541>.

[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>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

7.2. URIs

[1] https://github.com/mnot/I-D/labels/binary-structured-headers

[2] https://mnot.github.io/I-D/binary-structured-headers/

[3] https://github.com/mnot/I-D/commits/gh-pages/binary-structured-headers

[4] https://datatracker.ietf.org/doc/draft-nottingham-binary-structured-headers/

[5] https://github.com/mnot/I-D/issues/305

[6] https://github.com/mnot/I-D/issues/305

[7] https://github.com/mnot/I-D/issues/305

[8] https://github.com/mnot/I-D/issues/305

[9] https://github.com/mnot/I-D/issues/307

[10] https://github.com/mnot/I-D/issues/308

Author's Address

Mark Nottingham
Fastly

Email: mnot@mnot.net
URI: https://www.mnot.net/