Workgroup: Network Working Group
Internet-Draft:
draft-nottingham-binary-structured-headers-03
Published: 11 October 2022
Intended Status: Standards Track
Expires: 14 April 2023
Authors: M. Nottingham

Binary Structured HTTP Field Values

Abstract

This specification defines a binary serialisation of Structured Field Values for HTTP, along with a negotiation mechanism for its use in HTTP/2.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-nottingham-binary-structured-headers/.

information can be found at https://mnot.github.io/I-D/.

Source for this draft and an issue tracker can be found at https://github.com/mnot/I-D/labels/binary-structured-headers.

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 14 April 2023.

Table of Contents

1. Introduction

Structured Field Values for HTTP [STRUCTURED-FIELDS] offers a set of data types for use by HTTP fields, along with a serialisation of them in a familiar textual syntax.

Section 2 defines an alternative, binary serialisation of those structures, and Section 3 defines a mechanism for using that serialisation in HTTP/2.

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

This specification describes wire formats using the convention described in Section 1.3 of [QUIC].

2. Binary Structured Types

This section defines a binary serialisation for Structured Field Values as defined in [STRUCTURED-FIELDS].

A Structured Field Value can be a singular Item (such as a String, an Integer, or a Boolean, possibly with parameters), or it can be a compound type, such as a Dictionary or List, whose members are composed of those Item types.

When a field value is serialised as a Binary Structured Field, each of these types is preceded by an header octet that indicates the relevant type, along with some type-specific flags. The type then determines how the value is serialised in the following octet(s).

```
Binary Structured Type {
  Type (5),
  Flags (3),
  [Payload (..)]
}
```

Use of each flag may not be specified by all types. When this is the case, generators **MUST** send 0 for them, and recipients **MUST** ignore them.

2.1. Literal Values

A Literal Value is a special type that carries the string value of a field; they are used to carry field values that are not structured using the data types defined in Section 3 of [STRUCTURED-FIELDS]. This might be because the field is not recognised as a Structured Field, or it might be because a field that is understood to be a Structured Field cannot be parsed successfully as one.

A literal value's payload consists of an integer Length field (using the variable-length encoding from Section 16 of [QUIC]), followed by that many octets of the field value. They are functionally equivalent to String Literal Representations in Section 5.2 of [RFC7541].

```
Literal Value {
  Type (5) = 0,
  Unused Flags (3) = 0,
  Length (i),
  Payload (..)
}
```

2.2. Lists

A List (Section 3.1 of [STRUCTURED-FIELDS]) uses its Flags to carry a Short Member Count as three bits.

If the Short Member Count is zero, it indicates that the next byte is the Member Count, represented using the variable-length encoding from Section 16 of [QUIC]. Otherwise, it indicates the Member Count itself. This allows a small list to be encoded without using an extra byte for its length.

In either case, the payload indicates the number of List members that follow.

```
List {
  Type (5) = 1,
  Short Member Count (3),
  [ Member Count (i) ],
  Binary Structured Type (..) ...
}
```

2.3. Dictionaries

A Dictionary (Section 3.2 of [STRUCTURED-FIELDS]) uses its Flags to indicate its number of members, in a fashion similar to Lists.

```
Dictionary {
  Type (5) = 2,
  Short Member Count (3),
  [ Member Count (i) ],
  Dictionary Member (..) ...
}
```

Each Dictionary member is represented by a length, followed by that many bytes of the member-key, followed by the Binary Structured Type(s) representing the member-value.

```
Dictionary Member {
  Key Length (i),
  Member Key (..),
  Binary Structured Type (..),
}
```

A Dictionary Member **MUST NOT** be a Parameters (0x2).

2.4. Inner Lists

An Inner List (Section 3.1.1 of [STRUCTURED-FIELDS]) has a payload consisting a Member Count field (using the variable-length encoding from Section 16 of [QUIC]), followed by one or more fields representing the members of the list.

```
Inner List {
  Type (5) = 3,
  Parameters (1),
  Unused Flags (2),
  Member Count (i),
  Binary Structured Type (..) ...
  [Parameters (..)]
}
```

The Parameters Flag indicates whether the value is followed by Parameters (see Section 2.4.1).

Parameters on the Inner List itself, if present, are serialised in a following Parameter type (Section 2.4.1); they do not form part of the payload of the Inner List (and therefore are not counted in Member Count).

2.4.1. Parameters

Parameters (Section 3.1.2 of [STRUCTURED-FIELDS]) uses its Flags to indicate its number of members, in a fashion similar to Lists.

```
Parameters {
  Type (5) = 4,
  Short Parameter Count (3),
  [ Parameter Count (i) ],
  Parameter (..) ...
}
```

Each Parameter conveys a key and a value:

```
Parameter {
  Parameter Key Length (i),
  Parameter Key (..),
  Binary Structured Type (..)
}
```

A parameter's fields are:

- Parameter Key Length: The number of octets used for the parameter-key (using the variable-length encoding from Section 16 of [QUIC])

- Parameter Key: Parameter Key Length octets of the parameter-key

- Binary Structured Type: The parameter value

The Binary Structured Type in a Parameter **MUST NOT** be an Inner List (0x1) or Parameters (0x2).

Parameters are always associated with the Binary Structured Type that immediately preceded them. Therefore, Parameters **MUST NOT** be the first Binary Structured Type in a Binary Structured Field Value, and **MUST NOT** follow another Parameters.

2.4.2. Integers

An Integer (Section 3.3.1 of [STRUCTURED-FIELDS]) has a payload consisting of a single integer (using the variable-length encoding from Section 16 of [QUIC]). The Sign flag conveys whether the value is positive (1) or negative (0).

```
Integer {
  Type (5) = 5,
  Parameters (1),
  Sign (1),
  Unused Flag (1) = 0,
  Payload (i)
}
```

The Parameters Flag indicates whether the value is followed by Parameters (see Section 2.4.1).

2.4.3. Decimals

A Decimal (Section 3.3.2 of [STRUCTURED-FIELDS]) has a payload consisting of two integers (using the variable-length encoding from Section 16 of [QUIC]) that are divided to convey the decimal value. The Sign flag conveys whether the value is positive (1) or negative (0).

```
Decimal {
  Type (5) = 6,
  Parameters (1),
  Sign (1),
  Unused Flag (1) = 0,
  Dividend (i),
  Divisor (i)
}
```

The Parameters Flag indicates whether the value is followed by Parameters (see Section 2.4.1).

2.4.4. Strings

A String (Section 3.3.3 of [STRUCTURED-FIELDS]) has a payload consisting of an integer Length field (using the variable-length encoding from Section 16 of [QUIC]), followed by that many octets of payload.

```
String {
  Type (5) = 7,
  Parameters (1),
  Unused Flags (2) = 0,
  Length (i),
  Payload (..)
}
```

The Parameters Flag indicates whether the value is followed by Parameters (see Section 2.4.1).

Its payload is Length octets long and ASCII-encoded.

2.4.5. Tokens

A Token (Section 3.3.4 of [STRUCTURED-FIELDS]) has a payload consisting of an integer Length field (using the variable-length encoding from Section 16 of [QUIC]), followed by that many octets of payload.

```
Token {
  Type (5) = 8,
  Parameters (1),
  Unused Flags (2) = 0,
  Length (i),
  Payload (..)
}
```

The Parameters Flag indicates whether the value is followed by Parameters (see Section 2.4.1).

Its payload is Length octets long and ASCII-encoded.

2.4.6. Byte Sequences

A Byte Sequence (Section 3.3.5 of [STRUCTURED-FIELDS]) has a payload consisting of an integer Length field (using the variable-length encoding from Section 16 of [QUIC]), followed by that many octets of payload.

```
Byte Sequence {
  Type (5) = 9,
  Parameters (1),
  Unused Flags (2) = 0,
  Length (i),
  Payload (..)
}
```

The Parameters Flag indicates whether the value is followed by Parameters (see Section 2.4.1).

The payload is is Length octets long, containing the raw octets of the byte sequence.

2.4.7. Booleans

A String (Section 3.3.6 of [STRUCTURED-FIELDS]) uses the Payload flag to indicate its value; if Payload is 0, the value is False; if Payload is 1, the value is True.

The Boolean data type (type=0x8) carries its payload in the Payload Flag:

```
Boolean {
  Type (5) = 10,
  Parameters (1),
  Payload (1),
  Unused Flag (1) = 0,
}
```

The Parameters Flag indicates whether the value is followed by Parameters (see Section 2.4.1).

3. Using Binary Structured Fields in HTTP/2

When both peers on a connection support this specification, they can negotiate to serialise fields that they know to be Structured Fields as binary data, rather than strings.

Peers advertise and discover this support using a HTTP/2 setting defined in Section 3.1, and convey Binary Structured Fields in streams whose HEADERS frame uses the flag defined in Section 3.2.

3.1. The SETTINGS_BINARY_STRUCTURED_FIELDS Setting

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

Receiving SETTINGS_BINARY_STRUCTURED_FIELDS with a non-zero value from a peer indicates that:

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

2. The peer will process the BINARY_STRUCTRED HEADERS flag as defined in Section 3.2.

3. When passing the message to a downstream consumer (whether on the network or not) who does not support this extension or otherwise explicitly negotiate an equivalent mechanism, the peer will:

    1. Transform all fields defined as Mapped Fields in Section 1.3 of [RETROFIT] into their unmapped forms, removing the mapped fields.

    2. Serialize all fields into the appropriate form for that peer (e.g., the textual representation of Structured Fields data types defined in [STRUCTURED-FIELDS]).

The default value of SETTINGS_BINARY_STRUCTURED_FIELDS is 0, whereas a value of 1 indicates that this specification is supported with no further extensions. Future specifications might use values greater than one to indicate support for extensions.

3.2. The BINARY_STRUCTRED HEADERS Flag

When a peer has indicated that it supports this specification as per Section 3.1, a sender can send the BINARY_STRUCTURED flag (0xTODO) on the HEADERS frame.

This flag indicates that the HEADERS frame containing it and subsequent CONTINUATION frames on the same stream use the Binary Structured Types defined in Section 2 instead of the String Literal Representation defined in Section 5.2 of [RFC7541] to represent all field values. Field names are still serialised as String Literal Representations.

In such frames, all field values **MUST** be sent as Binary Structured Field Values. Note that this includes Binary Literals (Section 2.1) for those field values that are not recognised as Structured Fields, as well as textual values that cannot be successfully parsed as Structured Fields. Implementations **MAY** also send a field value as a Binary Literal even when it is possible to represent it as a Structured Field (e.g., for efficiency purposes).

Binary Structured Field Values are stored in the HPACK [RFC7541] dynamic table, and their lengths are used for the purposes of maintaining dynamic table size (see [RFC7541], Section 4).

Note that HEADERS frames with and without the BINARY_STRUCTURED flag **MAY** be mixed on the same connection, depending on the requirements of the sender.

4. IANA Considerations

*ISSUE: todo

5. 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 Fields data types, along with the "escape valve" of Binary Literals (Section 2.1). Therefore, implementation divergence from this strictness can have security impact.

6. Normative References

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RETROFIT] Nottingham, M., "Retrofit Structured Fields for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-retrofit-04, 8 June 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-retrofit-04>.

[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/rfc/rfc2119>.

[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/rfc/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/rfc/rfc8174>.

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

Author's Address

Mark Nottingham
Prahran
Australia

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