Abstract
This document defines a binary format for representing HTTP messages.

Status of This Memo
This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9292.

Copyright Notice
Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
Table of Contents

1. Introduction
2. Conventions and Definitions
3. Format
   3.1. Known-Length Messages
   3.2. Indeterminate-Length Messages
   3.3. Framing Indicator
   3.4. Request Control Data
   3.5. Response Control Data
      3.5.1. Informational Status Codes
   3.6. Header and Trailer Field Lines
   3.7. Content
   3.8. Padding and Truncation
4. Invalid Messages
5. Examples
   5.1. Request Example
   5.2. Response Example
6. Notable Differences with HTTP Protocol Messages
7. "message/bhttp" Media Type
8. Security Considerations
9. IANA Considerations
10. References
   10.1. Normative References
   10.2. Informative References

Acknowledgments
Authors' Addresses
1. Introduction

This document defines a simple format for representing an HTTP message [HTTP], either request or response. This allows for the encoding of HTTP messages that can be conveyed outside an HTTP protocol. This enables the transformation of entire messages, including the application of authenticated encryption.

The design of this format is informed by the framing structure of HTTP/2 [HTTP/2] and HTTP/3 [HTTP/3]. Rules for constructing messages rely on the rules defined in HTTP/2, but the format itself is distinct; see Section 6.

This format defines "message/bhttp", a binary alternative to the "message/http" content type defined in [HTTP/1.1]. A binary format permits more efficient encoding and processing of messages. A binary format also reduces exposure to security problems related to processing of HTTP messages.

Two modes for encoding are described:

- a known-length encoding includes length prefixes for all major message components, and
- an indeterminate-length encoding enables efficient generation of messages where lengths are not known when encoding starts.

This format is designed to convey the semantics of valid HTTP messages as simply and efficiently as possible. It is not designed to capture all of the details of the encoding of messages from specific HTTP versions [HTTP/1.1] [HTTP/2] [HTTP/3]. As such, this format is unlikely to be suitable for applications that depend on an exact recording of the encoding of messages.

2. Conventions and Definitions

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 document uses terminology from HTTP [HTTP] and notation from QUIC (Section 1.3 of [QUIC]).
3. Format

Section 6 of [HTTP] defines the general structure of HTTP messages and composes those messages into distinct parts. This format describes how those parts are composed into a sequence of bytes. At a high level, binary messages are comprised of:

1. Framing indicator. This format uses a single integer to describe framing, which describes whether the message is a request or response and how subsequent sections are formatted; see Section 3.3.
2. For a response, zero or more informational responses. Each informational response consists of an informational status code and header section.
3. Control data. For a request, this contains the request method and target. For a response, this contains the status code.
4. Header section. This contains zero or more header fields.
5. Content. This is a sequence of zero or more bytes.
6. Trailer section. This contains zero or more trailer fields.

All lengths and numeric values are encoded using the variable-length integer encoding from Section 16 of [QUIC]. Integer values do not need to be encoded on the minimum number of bytes necessary.

3.1. Known-Length Messages

A request or response that has a known length at the time of construction uses the format shown in Figure 1.
A known-length request consists of a framing indicator (Section 3.3), request control data (Section 3.4), a header section with a length prefix, binary content with a length prefix, a trailer section with a length prefix, and padding.

A known-length response contains the same fields, with the exception that request control data is replaced by zero or more informational responses (Section 3.5.1) followed by response control data (Section 3.5).

For a known-length encoding, the length prefix on field sections and content is a variable-length encoding of an integer. This integer is the number of bytes in the field section or content, not including the length field itself.

Fields in the header and trailer sections consist of a length-prefixed name and length-prefixed value; see Section 3.6.

The format allows for the message to be truncated before any of the length prefixes that precede the field sections or content; see Section 3.8.
The variable-length integer encoding means that there is a limit of $2^{62} - 1$ bytes for each field section and the message content.

### 3.2. Indeterminate-Length Messages

A request or response that is constructed without encoding a known length for each section uses the format shown in Figure 2:

```plaintext
Indeterminate-Length Request  {
  Framing Indicator (i) = 2,
  Request Control Data (..),
  Indeterminate-Length Field Section (..),
  Indeterminate-Length Content (..),
  Indeterminate-Length Field Section (..),
  Padding (..),
}

Indeterminate-Length Response  {
  Framing Indicator (i) = 3,
  Indeterminate-Length Informational Response (..) ..., 
  Final Response Control Data (..),
  Indeterminate-Length Field Section (..),
  Indeterminate-Length Content (..),
  Indeterminate-Length Field Section (..),
  Padding (..),
}

Indeterminate-Length Content  {
  Indeterminate-Length Content Chunk (..) ..., 
  Content Terminator (i) = 0,
}

Indeterminate-Length Content Chunk  {
  Chunk Length (i) = 1..,
  Chunk (..),
}

Indeterminate-Length Field Section  {
  Field Line (..) ..., 
  Content Terminator (i) = 0,
}

Indeterminate-Length Informational Response  {
  Informational Response Control Data (..), 
  Indeterminate-Length Field Section (..),
}
```

*Figure 2: Indeterminate-Length Message*

An indeterminate-length request consists of a framing indicator (Section 3.3), request control data (Section 3.4), a header section that is terminated by a zero value, any number of non-zero-length chunks of binary content, a zero value, a trailer section that is terminated by a zero value, and padding.
An indeterminate-length response contains the same fields, with the exception that request control data is replaced by zero or more informational responses (Section 3.5.1) and response control data (Section 3.5).

The indeterminate-length encoding only uses length prefixes for content blocks. Multiple length-prefixed portions of content can be included, each prefixed by a non-zero Chunk Length integer describing the number of bytes in the block. The Chunk Length is encoded as a variable-length integer.

Each Field Line in an Indeterminate-Length Field Section starts with a Name Length field. An Indeterminate-Length Field Section ends with a Content Terminator field. The zero value of the Content Terminator distinguishes it from the Name Length field, which cannot contain a value of 0.

Indeterminate-length messages can be truncated in a way similar to that for known-length messages; see Section 3.8.

Indeterminate-length messages use the same encoding for Field Line as known-length messages; see Section 3.6.

### 3.3. Framing Indicator

The start of each binary message is a framing indicator that is a single integer that describes the structure of the subsequent sections. The framing indicator can take just four values:

- A value of 0 describes a request of known length.
- A value of 1 describes a response of known length.
- A value of 2 describes a request of indeterminate length.
- A value of 3 describes a response of indeterminate length.

Other values cause the message to be invalid; see Section 4.

### 3.4. Request Control Data

The control data for a request message contains the method and request target. That information is encoded as an ordered sequence of fields: Method, Scheme, Authority, Path. Each of these fields is prefixed with a length.

The values of these fields follow the rules in HTTP/2 (Section 8.3.1 of [HTTP/2]) that apply to the ":method", ":scheme", ":authority", and ":path" pseudo-header fields, respectively. However, where the ":authority" pseudo-header field might be omitted in HTTP/2, a zero-length value is encoded instead.

The format of request control data is shown in Figure 3.
3.5. Response Control Data

The control data for a response message consists of the status code. The status code (Section 15 of [HTTP]) is encoded as a variable-length integer, not a length-prefixed decimal string.

The format of final response control data is shown in Figure 4.

```
Final Response Control Data {
   Status Code (i) = 200..599,
}
```

Figure 4: Format of Final Response Control Data

3.5.1. Informational Status Codes

Responses that include informational status codes (see Section 15.2 of [HTTP]) are encoded by repeating the response control data and associated header section until a final status code is encoded; that is, a Status Code field with a value from 200 to 599 (inclusive). The status code distinguishes between informational and final responses.

The format of the informational response control data is shown in Figure 5.

```
Informational Response Control Data {
   Status Code (i) = 100..199,
}
```

Figure 5: Format of Informational Response Control Data

A response message can include any number of informational responses that precede a final status code. These convey an informational status code and a header block.

If the response control data includes an informational status code (that is, a value between 100 and 199 inclusive), the control data is followed by a header section (encoded with known length or indeterminate length according to the framing indicator) and another block of control data. This pattern repeats until the control data contains a final status code (200 to 599 inclusive).
3.6. Header and Trailer Field Lines

Header and trailer sections consist of zero or more field lines; see Section 5 of [HTTP]. The format of a field section depends on whether the message is of known length or indeterminate length.

Each Field Line encoding includes a name and a value. Both the name and value are length-prefixed sequences of bytes. The Name field is a minimum of one byte. The format of a Field Line is shown in Figure 6.

For field names, byte values that are not permitted in an HTTP field name cause the message to be invalid; see Section 5.1 of [HTTP] for a definition of what is valid and Section 4 regarding the handling of invalid messages. A recipient MUST treat a message that contains field values that would cause an HTTP/2 message to be malformed according to Section 8.2.1 of [HTTP/2] as invalid; see Section 4.

The same field name can be repeated over more than one field line; see Section 5.2 of [HTTP] for the semantics of repeated field names and rules for combining values.

Messages are invalid (Section 4) if they contain fields named "::method", "::scheme", "::authority", "::path", or "::status". Other pseudo-fields that are defined by protocol extensions MAY be included; pseudo-fields cannot be included in trailers (see Section 8.1 of [HTTP/2]). A Field Line containing pseudo-fields MUST precede other Field Line values. A message that contains a pseudo-field after any other field is invalid; see Section 4.

Fields that relate to connections (Section 7.6.1 of [HTTP]) cannot be used to produce the effect on a connection in this context. These fields SHOULD be removed when constructing a binary message. However, they do not cause a message to be invalid (Section 4); permitting these fields allows a binary message to capture messages that are exchanged in a protocol context.

Like HTTP/2 or HTTP/3, this format has an exception for the combination of multiple instances of the Cookie field. Instances of fields with the ASCII-encoded value of "cookie" are combined using a semicolon octet (0x3b) rather than a comma; see Section 8.2.3 of [HTTP/2].

3.7. Content

The content of messages is a sequence of bytes of any length. Though a known-length message has a limit, this limit is large enough that it is unlikely to be a practical limitation. There is no limit to the size of content in an indeterminate-length message.
3.8. Padding and Truncation

Messages can be padded with any number of zero-valued bytes. Non-zero padding bytes cause a message to be invalid (see Section 4). Unlike other parts of a message, a processor MAY decide not to validate the value of padding bytes.

Truncation can be used to reduce the size of messages that have no data in trailing field sections or content. If the trailers of a message are empty, they MAY be omitted by the encoder in place of adding a length field equal to zero. An encoder MAY omit empty content in the same way if the trailers are also empty. A message that is truncated at any other point is invalid; see Section 4.

Decoders MUST treat missing truncated fields as equivalent to having been sent with the length field set to zero.

Padding is compatible with truncation of empty parts of the messages. Zero-valued bytes will be interpreted as a zero-length part, which is semantically equivalent to the part being absent.

4. Invalid Messages

This document describes a number of ways that a message can be invalid. Invalid messages MUST NOT be processed further except to log an error and produce an error response.

The format is designed to allow incremental processing. Implementations need to be aware of the possibility that an error might be detected after performing incremental processing.

5. Examples

This section includes example requests and responses encoded in both known-length and indeterminate-length forms.

5.1. Request Example

The example HTTP/1.1 message in Figure 7 shows the content in the "message/http" format.

Valid HTTP/1.1 messages require lines terminated with CRLF (the two bytes 0x0d and 0x0a). For simplicity and consistency, the content of these examples is limited to text, which also uses CRLF for line endings.

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

*Figure 7: Sample HTTP Request*
This can be expressed as a binary message (type "message/bhttp") using a known-length encoding as shown in hexadecimal in Figure 8. Figure 8 includes text alongside to show that most of the content is not modified.

```
00034745 54056874 74707300 0a2f6865 ..GET.https../he
6c6f6f2e 746c6c6f2e 7478740a 75736572  llo.txt@l.user-a
67766e64 34637572 302e332e 376c207a   gent4curl/7.16.3
6c6f6f2e 6c6f6f2e 372e3136 206c6f6f6b  libcurl/7.16.3
776f726c 7365726e 756167652d6c616e67  w.example.com.a
6e2c206d 690000                    n, mi.
```

Figure 8: Known-Length Binary Encoding of Request

This example shows that the Host header field is not replicated in the ":authority" field, as is required for ensuring that the request is reproduced accurately; see Section 8.3.1 of [HTTP/2].

The same message can be truncated with no effect on interpretation. In this case, the last two bytes -- corresponding to content and a trailer section -- can each be removed without altering the semantics of the message.

The same message, encoded using an indeterminate-length encoding, is shown in Figure 9. As the content of this message is empty, the difference in formats is negligible.

```
02034745 54056874 74707300 0a2f6865 ..GET.https../he
6c6f6f2e 7478740a 75736572 2d616765 llo.txt.user-age
6e734663 75726c2f 372e3136 2e332e30  nt4curl/7.16.3 l
69626c75 726c2f37 2e31362e 33284f70 libcurl/7.16.3 Op
656e5353 4c2f302e 302e392e 7a6c69622f OpenSSL/0.9.7l zli
622f312e 332e3004 686f7374 0f777777 b/1.2.3.host.www
2e6f7861 6d786e65 6e656e6c2d686f73 ept-language.en,
206d6900 00000000 00000000 000000 mi...........
```

Figure 9: Indeterminate-Length Binary Encoding of Request

This indeterminate-length encoding contains 10 bytes of padding. As two additional bytes can be truncated in the same way as the known-length example, anything up to 12 bytes can be removed from this message without affecting its meaning.

5.2. Response Example

Response messages can contain interim (1xx) status codes, as the message in Figure 10 shows. Figure 10 includes examples of informational status codes 102 and 103, as defined in [RFC2518] (now obsolete but defines status code 102) and [RFC8297], respectively.
As this is a longer example, only the indeterminate-length encoding is shown in Figure 11. Note here that the specific text used in the reason phrase is not retained by this encoding.

Hello World! My content includes a trailing CRLF.

Figure 10: Sample HTTP Response

Figure 11: Binary Response, including Informational Responses
A response that uses the chunked encoding (see Section 7.1 of [HTTP/1.1]) as shown in Figure 12 can be encoded using indeterminate-length encoding, which minimizes buffering needed to translate into the binary format. However, chunk boundaries do not need to be retained, and any chunk extensions cannot be conveyed using the binary format; see Section 6.

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked

4
This
6
content
13;chunk-extension=foo
nt contains CRLF.

0
Trailer: text
```

Figure 12: Chunked Encoding Example

Figure 13 shows this message using the known-length encoding. Note that the Transfer-Encoding header field is removed.

```
0140c800 1d546869 7320636f 6e74656e  .@...This conten
7420636f 6e746169 6e732043 524c462e  t contains CRLF.
0d0a0d07 74726169 6c657204 74657874  ....trailer.text
```

Figure 13: Known-Length Encoding of Response

6. Notable Differences with HTTP Protocol Messages

This format is designed to carry HTTP semantics just like HTTP/1.1 [HTTP/1.1], HTTP/2 [HTTP/2], or HTTP/3 [HTTP/3]. However, there are some notable differences between this format and the format used in an interactive protocol version.

In particular, as a standalone representation, this format lacks the following features of the formats used in those protocols:

- chunk extensions (Section 7.1.1 of [HTTP/1.1]) and transfer encoding (Section 6.1 of [HTTP/1.1])
- generic framing and extensibility capabilities
- field blocks other than a single header and trailer field block
- carrying reason phrases in responses (Section 4 of [HTTP/1.1])
- header compression [HPACK] [QPACK]
response framing that depends on the corresponding request (such as HEAD) or the value of the status code (such as 204 or 304); these responses use the same framing as all other messages.

Some of these features are also absent in HTTP/2 and HTTP/3.

Unlike HTTP/2 and HTTP/3, this format uses a fixed format for control data rather than using pseudo-fields.

Note that while some messages -- CONNECT or upgrade requests in particular -- can be represented using this format, doing so serves no purpose, as these requests are used to affect protocol behavior, which this format cannot do without additional mechanisms.

7. "message/bhttp" Media Type

The "message/bhttp" media type can be used to enclose a single HTTP request or response message, provided that it obeys the MIME restrictions for all "message" types regarding line length and encodings.

Type name: message
Subtype name: bhttp
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: Only "8bit" or "binary" is permitted.
Security considerations: See Section 8.
Interoperability considerations: N/A
Published specification: RFC 9292
Applications that use this media type: Applications seeking to convey HTTP semantics that are independent of a specific protocol.
Fragment identifier considerations: N/A
Additional information: Deprecated alias names for this type: N/A
   Magic number(s): N/A
   File extension(s): N/A
   Macintosh file type code(s): N/A
Person & email address to contact for further information: See the Authors' Addresses section.
Intended usage: COMMON
Restrictions on usage: N/A
8. Security Considerations

Many of the considerations that apply to HTTP message handling apply to this format; see Section 17 of [HTTP] and Section 11 of [HTTP/1.1] for common issues in handling HTTP messages.

Strict parsing of the format with no tolerance for errors can help avoid a number of attacks. However, implementations still need to be aware of the possibility of resource exhaustion attacks that might arise from receiving large messages, particularly those with large numbers of fields.

Implementations need to ensure that they aren’t subject to resource exhaustion attacks from maliciously crafted messages. Overall, the format is designed to allow for minimal state when processing messages. However, producing a combined field value (Section 5.2 of [HTTP]) for fields might require the commitment of resources. In particular, combining might be necessary for the Cookie field when translating this format for use in other contexts, such as use in an API or translation to HTTP/1.1 [HTTP/1.1], where the recipient of the field might not expect multiple values.

9. IANA Considerations

IANA has added the media type "message/bhttp" to the "Media Types" registry at <https://www.iana.org/assignments/media-types>. See Section 7 for registration information.

10. References

10.1. Normative References


Acknowledgments

Julian Reschke, David Schinazi, Lucas Pardue, and Tommy Pauly provided excellent feedback on both the design and its documentation.

Authors' Addresses

Martin Thomson
Mozilla
Email: mt@lowentropy.net

Christopher A. Wood
Cloudflare
Email: caw@heapingbits.net