

HTTPbis Working Group
Internet-Draft
Intended status: Informational
Expires: June 7, 2014

R. Peon
Google, Inc
H. Ruellan
Canon CRF
December 4, 2013

HPACK - Header Compression for HTTP/2.0
draft-ietf-httpbis-header-compression-05

Abstract

This document describes HPACK, a format adapted to efficiently represent HTTP header fields in the context of HTTP/2.0.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

Working Group information and related documents can be found at <http://tools.ietf.org/wg/httpbis/> (Wiki) and <https://github.com/http2/http2-spec> (source code and issues tracker).

The changes in this draft are summarized in [Appendix A.1](#).

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 June 7, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the

Internet-Draft

HPACK

December 2013

document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	4
2.	Overview	4
2.1.	Outline	4
3.	Header Field Encoding	5
3.1.	Encoding Concepts	5
3.1.1.	Encoding Context	5
3.1.2.	Header Table	6
3.1.3.	Reference Set	6
3.1.4.	Header Field Representation	7
3.1.5.	Header Field Emission	8
3.2.	Header Block Decoding	8
3.2.1.	Header Field Representation Processing	8
3.2.2.	Reference Set Emission	10
3.2.3.	Header Set Completion	10
3.3.	Header Table Management	10
3.3.1.	Maximum Table Size	10
3.3.2.	Entry Eviction When Header Table Size Changes	10
3.3.3.	Entry Eviction when Adding New Entries	11
4.	Detailed Format	11
4.1.	Low-level representations	11
4.1.1.	Integer representation	11
4.1.2.	String Literal Representation	13
4.2.	Indexed Header Field Representation	15
4.3.	Literal Header Field Representation	15
4.3.1.	Literal Header Field without Indexing	15
4.3.2.	Literal Header Field with Incremental Indexing	16
5.	Security Considerations	18
6.	References	18
6.1.	Normative References	18

6.2.	Informative References	19
Appendix A.	Change Log (to be removed by RFC Editor before publication)	19
A.1.	Since draft-ietf-httpbis-header-compression-04	19
A.2.	Since draft-ietf-httpbis-header-compression-03	20

A.3.	Since draft-ietf-httpbis-header-compression-02	20
A.4.	Since draft-ietf-httpbis-header-compression-01	20
A.5.	Since draft-ietf-httpbis-header-compression-01	21
Appendix B.	Static Table	21
Appendix C.	Huffman Codes For Requests	23
Appendix D.	Huffman Codes for Responses	29
Appendix E.	Examples	34
E.1.	Header Field Representation Examples	34
E.1.1.	Literal Header Field with Indexing	34
E.1.2.	Literal Header Field without Indexing	35
E.1.3.	Indexed Header Field	36
E.1.4.	Indexed Header Field from Static Table	37
E.2.	Request Examples without Huffman	37
E.2.1.	First request	37
E.2.2.	Second request	38
E.2.3.	Third request	40
E.3.	Request Examples with Huffman	42
E.3.1.	First request	42
E.3.2.	Second request	43
E.3.3.	Third request	44
E.4.	Response Examples without Huffman	46
E.4.1.	First response	46
E.4.2.	Second response	48
E.4.3.	Third response	49
E.5.	Response Examples with Huffman	51
E.5.1.	First response	51
E.5.2.	Second response	53
E.5.3.	Third response	54

1. Introduction

This document describes HPACK, a format adapted to efficiently represent HTTP header fields in the context of HTTP/2.0 (see [\[HTTP2\]](#)).

2. Overview

In HTTP (see [\[HTTP-p1\]](#)), header fields are sent without any form of compression. As web pages have grown to include dozens to hundreds of requests, the redundant header fields in these requests now pose a problem of measurable latency and unnecessary bandwidth (see [\[PERF1\]](#) and [\[PERF2\]](#)).

SPDY [\[SPDY\]](#) initially addressed this redundancy by compressing header fields with Deflate, which proved very effective at eliminating the redundant header fields. However, that approach exposed a security risk as demonstrated by the CRIME [\[CRIME\]](#).

This document describes HPACK, a new compressor for header fields which eliminates redundant header fields, is not vulnerable to known security attacks, and which also has a bounded memory cost for use in constrained environments.

2.1. Outline

The HTTP header field encoding described in this document is based on a header table that map name-value pairs to index values. Header tables are incrementally updated during the HTTP/2.0 session.

The encoder is responsible for deciding which header fields to insert as new entries in the header table. The decoder then does exactly what the encoder prescribes, ending in a state that exactly matches the encoder's state. This enables decoders to remain simple and understand a wide variety of encoders.

As two consecutive sets of header fields often have header fields in common, each set of header fields is coded as a difference from the previous set of header fields. The goal is to only encode the changes (header fields present in one of the set and not in the other) between the two sets of header fields.

HTTP header field compression treats a set of header fields as an unordered collection of name-value pairs. Names and values are opaque sequences of octets. The order of header fields is not guaranteed to be preserved after being compression and decompression.

Examples illustrating the use of these different mechanisms to

represent header fields are available in [Appendix E](#).

[3.](#) Header Field Encoding

[3.1.](#) Encoding Concepts

The encoding and decoding of header fields relies on some components and concepts:

Header Field: A name-value pair. Both name and value are sequences of octets.

Header Table: The header table (see [Section 3.1.2](#)) is a component used to associate stored header fields to index values. The data stored in this table is in first-in, first-out order.

Static Table: The static table (see [Appendix B](#)) is a component used to associate static header fields to index values. This data is ordered, read-only, always accessible, and may be shared amongst all encoding contexts.

Reference Set: The reference set (see [Section 3.1.3](#)) is a component

containing an unordered set of references to entries in the header table. This is used for the differential encoding of a new header set.

Header Set: A header set is a potentially ordered group of header fields that are encoded jointly. A complete set of key-value pairs contained in a HTTP request or response is a header set.

Header Field Representation: A header field can be represented in encoded form either as a literal or as an index (see [Section 3.1.4](#)).

Header Block: The entire set of encoded header field representations which, when decoded, yield a complete header set.

Header Field Emission: When decoding a set of header field representations, some operations emit a header field (see [Section 3.1.5](#)). Emitted header fields can be safely passed to the upper processing layers as part of the current Header Set.

[3.1.1.](#) Encoding Context

The set of mutable structures used within an encoding context include a header table and a reference set. Everything else is either immutable or conceptual.

Using HTTP, messages are exchanged between a client and a server in both direction. To keep the encoding of header fields in each direction independent from the other direction, there is one encoding context for each direction.

The header fields contained in a PUSH_PROMISE frame sent by a server to a client are encoded within the same context as the header fields contained in the HEADERS frame corresponding to a response sent from the server to the client.

[3.1.2.](#) Header Table

A header table consists of a list of header fields maintained in first-in, first-out order. The first and newest entry in a header table is always at index 1, and the oldest entry of a header table is

at the index `len(header table)`.

The header table is initially empty.

There is typically no need for the header table to contain duplicate entries. However, duplicate entries **MUST NOT** be treated as an error by a decoder.

The encoder decides how to update the header table and as such can control how much memory is used by the header table. To limit the memory requirements on the decoder side, the header table size is strictly bounded (see [Section 3.3.1](#)).

The header table is updated during the processing of a set of header field representations (see header field representation processing ([Section 3.2.1](#))).

[3.1.3](#). Reference Set

A reference set is an unordered set of references to entries of the header table.

The reference set is initially empty.

The reference set is updated during the processing of a set of header field representations (see header field representation processing ([Section 3.2.1](#))).

The reference set enables differential encoding, whereby only differences between the previous header set and the current header set need to be encoded. The use of differential encoding is optional for any header set.

When an entry is evicted from the header table, if it was referenced from the reference set, its reference is removed from the reference set.

To limit the memory requirements on the decoder side for handling the reference set, only entries within the header table can be contained in the reference set. To still allow entries from the static table to take advantage of the differential encoding, when a header field

is represented as a reference to an entry of the static table, this entry is inserted into the header table ((see [Section 3.2.1](#)).

3.1.4. Header Field Representation

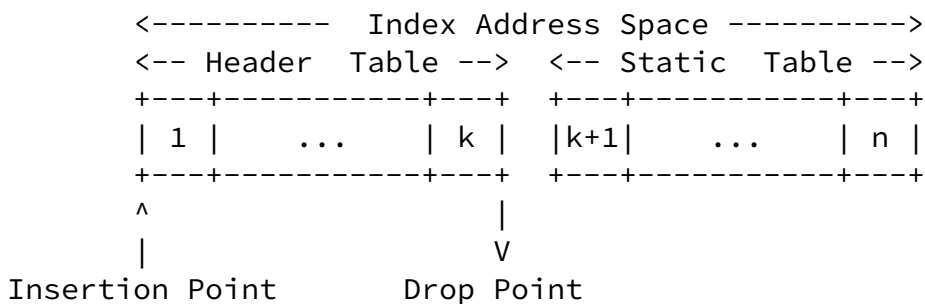
An encoded header field can be represented either as a literal or as an index.

Literal Representation: A literal representation defines a new header field. The header field name is represented either literally or as a reference to an entry of the header table. The header field value is represented literally.

Two different literal representations are provided:

- * A literal representation that does not add the header field to the header table (see [Section 4.3.1](#)).
- * A literal representation that adds the header field as a new entry at the beginning of the header table (see [Section 4.3.2](#)).

Indexed Representation: The indexed representation defines a header field as a reference to an entry in either the header table or the static table(see [Section 4.2](#)).



Index Address Space

elements in the header table, with index 1 referring to the beginning of the table.

Indices between $\text{len}(\text{header table})+1$ and $\text{len}(\text{header table})+\text{len}(\text{static table})$, inclusive, refer to elements in the static table, where the index $\text{len}(\text{header table})+1$ refers to the first entry in the static table.

Index 0 signals that the reference set MUST be emptied.

Any other indices MUST be treated as erroneous, and the compression context considered corrupt and unusable.

[3.1.5.](#) Header Field Emission

The emission of a header field is the process of marking a header field as belonging to the current header set. Once a header has been emitted, it cannot be removed from the current header set.

On the decoding side, an emitted header field can be safely passed to the upper processing layer as part of the current header set. The decoder MAY pass the emitted header fields to the upper processing layer in any order.

By emitting header fields instead of emitting header sets, the decoder can be implemented in a streaming way, and as such has only to keep in memory the header table and the reference set. This bounds the amount of memory used by the decoder, even in presence of a very large set of header fields. The management of memory for handling very large sets of header fields can therefore be deferred to the upper processing layers.

[3.2.](#) Header Block Decoding

The processing of a header block to obtain a header set is defined in this section. To ensure that the decoding will successfully produce a header set, a decoder MUST obey the following rules.

[3.2.1.](#) Header Field Representation Processing

All the header field representations contained in a header block are processed in the order in which they are presented, as specified below.

An `_indexed representation_` with an index value of 0 entails the following actions:

- o The reference set is emptied.

An `_indexed representation_` corresponding to an entry `_present_` in the reference set entails the following actions:

- o The entry is removed from the reference set.

An `_indexed representation_` corresponding to an entry `_not present_` in the reference set entails the following actions:

- o If referencing an element of the static table:
 - * The header field corresponding to the referenced entry is emitted.
 - * The referenced static entry is inserted at the beginning of the header table.
 - * A reference to this new header table entry is added to the reference set (except if this new entry didn't fit in the header table).
- o If referencing an element of the header table:
 - * The header field corresponding to the referenced entry is emitted.
 - * The referenced header table entry is added to the reference set.

A `_literal representation_` that is `_not added_` to the header table entails the following action:

- o The header field is emitted.

A `_literal representation_` that is `_added_` to the header table entails the following actions:

- o The header field is emitted.
- o The header field is inserted at the beginning of the header table.
- o A reference to the new entry is added to the reference set (except if this new entry didn't fit in the header table).

[3.2.2.](#) Reference Set Emission

Once all the representations contained in a header block have been processed, the header fields referenced in the reference set which have not previously been emitted during this processing are emitted.

[3.2.3.](#) Header Set Completion

Once all of the header field representations have been processed, and the remaining items in the reference set have been emitted, the header set is complete.

[3.3.](#) Header Table Management

[3.3.1.](#) Maximum Table Size

To limit the memory requirements on the decoder side, the size of the header table is bounded. The size of the header table MUST stay lower than or equal to the value of the HTTP/2.0 setting `SETTINGS_HEADER_TABLE_SIZE` (see [[HTTP2](#)]).

The size of the header table is the sum of the size of its entries.

The size of an entry is the sum of its name's length in octets (as defined in [Section 4.1.2](#)), of its value's length in octets ([Section 4.1.2](#)) and of 32 octets.

The lengths are measured on the non-encoded entry name and entry value (for the case when a Huffman encoding is used to transmit string values).

The 32 octets are an accounting for the entry structure overhead. For example, an entry structure using two 64-bits pointers to reference the name and the value and the entry, and two 64-bits integer for counting the number of references to these name and value would use 32 octets.

[3.3.2.](#) Entry Eviction When Header Table Size Changes

Whenever an entry is evicted from the header table, any reference to that entry contained by the reference set is removed.

Whenever `SETTINGS_HEADER_TABLE_SIZE` is made smaller, entries are evicted from the end of the header table until the size of the header table is less than or equal to `SETTINGS_HEADER_TABLE_SIZE`.

The eviction of an entry from the header table causes the index of the entries in the static table to be reduced by one.

[3.3.3.](#) Entry Eviction when Adding New Entries

Whenever a new entry is to be added to the table, any name referenced by the representation of this new entry is cached, and then entries are evicted from the end of the header table until the size of the header table is less than or equal to `SETTINGS_HEADER_TABLE_SIZE - new entry size`, or until the table is empty.

If the size of the new entry is less than or equal to `SETTINGS_HEADER_TABLE_SIZE`, that entry is added to the table. It is not an error to attempt to add an entry that is larger than `SETTINGS_HEADER_TABLE_SIZE`.

[4.](#) Detailed Format

[4.1.](#) Low-level representations

[4.1.1.](#) Integer representation

Integers are used to represent name indexes, pair indexes or string lengths. To allow for optimized processing, an integer representation always finishes at the end of an octet.

An integer is represented in two parts: a prefix that fills the current octet and an optional list of octets that are used if the integer value does not fit within the prefix. The number of bits of the prefix (called *N*) is a parameter of the integer representation.

The *N*-bit prefix allows filling the current octet. If the value is small enough (strictly less than 2^N-1), it is encoded within the *N*-bit prefix. Otherwise all the bits of the prefix are set to 1 and the value is encoded using an unsigned variable length integer [[1](#)]

representation. N is always between 1 and 8 bits. An integer starting at an octet-boundary will have an 8-bit prefix.

The algorithm to represent an integer I is as follows:

```
If I < 2^N - 1, encode I on N bits
Else
  encode 2^N - 1 on N bits
  I = I - (2^N - 1)
  While I >= 128
    Encode (I % 128 + 128) on 8 bits
    I = I / 128
  encode (I) on 8 bits
```

This integer representation allows for values of indefinite size. It is also possible for an encoder to send a large number of zero

values, which can waste octets and could be used to overflow integer values. Excessively large integer encodings - in value or octet length - MUST be treated as a decoding error. Different limits can be set for each of the different uses of integers, based on implementation constraints.

[4.1.1.1](#). Example 1: Encoding 10 using a 5-bit prefix

The value 10 is to be encoded with a 5-bit prefix.

- o 10 is less than 31 ($= 2^5 - 1$) and is represented using the 5-bit prefix.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| X | X | X | 0 | 1 | 0 | 1 | 0 |   10 stored on 5 bits
+---+---+---+---+---+---+---+---+
```

[4.1.1.2](#). Example 2: Encoding 1337 using a 5-bit prefix

The value I=1337 is to be encoded with a 5-bit prefix.

1337 is greater than 31 ($= 2^5 - 1$).

The 5-bit prefix is filled with its max value (31).

$$I = 1337 - (2^5 - 1) = 1306.$$

I (1306) is greater than or equal to 128, the while loop body executes:

$$I \% 128 == 26$$

$$26 + 128 == 154$$

154 is encoded in 8 bits as: 10011010

$$I \text{ is set to } 10 \text{ (} 1306 / 128 == 10 \text{)}$$

I is no longer greater than or equal to 128, the while loop terminates.

I, now 10, is encoded on 8 bits as: 00001010

The process ends.

0	1	2	3	4	5	6	7										
+	+	+	+	+	+	+	+	+									
	X		X		X		1		1		1		1		1		Prefix = 31, I = 1306
	1		0		0		1		1		0		1		0		1306 >= 128, encode(154), I=1306/128
	0		0		0		0		1		0		1		0		10 < 128, encode(10), done
+	+	+	+	+	+	+	+	+									

[4.1.1.3](#). Example 3: Encoding 42 starting at an octet-boundary

The value 42 is to be encoded starting at an octet-boundary. This implies that a 8-bit prefix is used.

- o 42 is less than 255 ($= 2^8 - 1$) and is represented using the 8-bit prefix.

```
    0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |   42 stored on 8 bits
+---+---+---+---+---+---+---+---+
```

[4.1.2.](#) String Literal Representation

Header field names and header field values are encoded as sequences of octets. A header field name or a header field value is encoded in three parts:

1. One bit, H, indicating whether or not the octets are Huffman encoded.
2. The number of octets required to hold the result of the next step, represented as a variable-length-quantity ([Section 4.1.1](#)), starting with a 7-bit prefix immediately following the first bit.
3. The encoded data of the string:
 1. If H is true, then the encoded string data is the bitwise concatenation of the canonical [[CANON](#)] Huffman code [[HUFF](#)] corresponding to each octet of the data, followed by between 0-7 bits of padding.

2. If H is false, then the encoded string is the octets of the field value without modification.

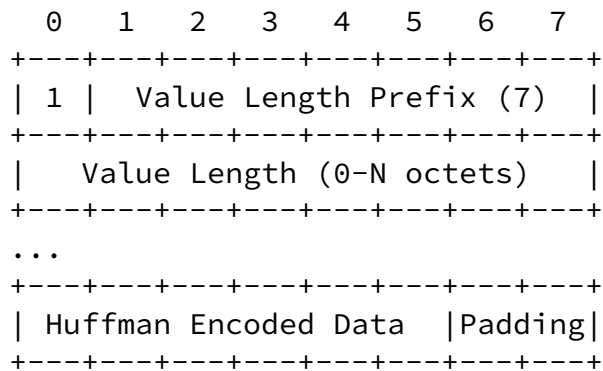
Padding is necessary when doing Huffman encoding to ensure that the remaining bits between the actual end of the data and the next octet boundary are not misinterpreted as part of the input data.

When padding for Huffman encoding, use the bits from the EOS (end-of-string) entry in the Huffman table, starting with the MSB (most significant bit). This entry is guaranteed to be at least 8 bits long.

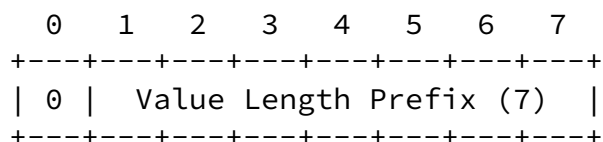
String literals sent in the client to server direction which use Huffman encoding are encoded with the codes within the request Huffman code table (Appendix C) (see Request Examples With Huffman (Appendix E.3)).

String literals sent in the server to client direction which use Huffman encoding are encoded with the codes within the response Huffman code table (Appendix D) (see Response Examples With Huffman (Appendix E.5)).

The EOS symbol is represented with value 256, and is used solely to signal the end of the Huffman-encoded key data or the end of the Huffman-encoded value data. Given that only between 0-7 bits of the EOS symbol is included in any Huffman-encoded string, and given that the EOS symbol is at least 8 bits long, it is expected that it should never be successfully decoded.



String Literal With Huffman Encoding




```

| Value Length (0-N octets) |
+---+---+---+---+---+---+---+---+
...
+---+---+---+---+---+---+---+---+
| Field Bytes Without Encoding |
+---+---+---+---+---+---+---+---+

```

String Literal Without Huffman Encoding

4.2. Indexed Header Field Representation

An indexed header field representation either identifies an entry in the header table or static table. The processing of an indexed header field representation is described in [Section 3.2.1](#).

```

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

```

Indexed Header Field

This representation starts with the '1' 1-bit pattern, followed by the index of the matching pair, represented as an integer with a 7-bit prefix.

The index value of 0 is reserved for signalling that the reference set is emptied.

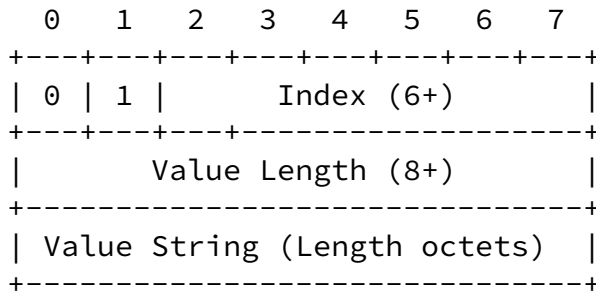
4.3. Literal Header Field Representation

Literal header field representations contain a literal header field value. Header field names are either provided as a literal or by reference to an existing header table or static table entry.

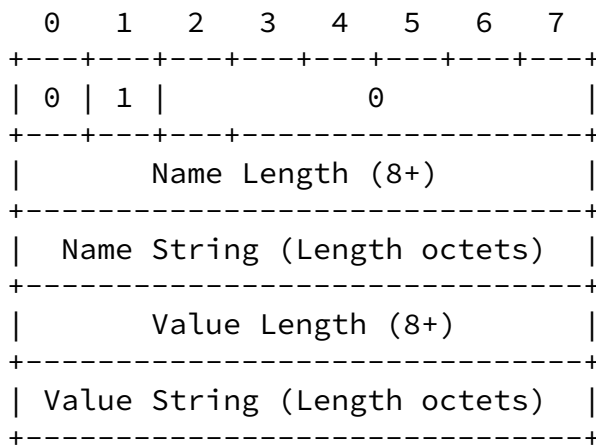
Literal representations all result in the emission of a header field when decoded.

4.3.1. Literal Header Field without Indexing

A literal header field without indexing causes the emission of a header field without altering the header table.



Literal Header Field without Indexing - Indexed Name



Literal Header Field without Indexing - New Name

This representation starts with the '01' 2-bit pattern.

If the header field name matches the header field name of a (name, value) pair stored in the Header Table or Static Table, the header field name can be represented using the index of that entry. In this case, the index of the entry, index (which is strictly greater than 0), is represented as an integer with a 6-bit prefix (see [Section 4.1.1](#)).

Otherwise, the header field name is represented as a literal. The value 0 is represented on 6 bits followed by the header field name (see [Section 4.1.2](#)).

The header field name representation is followed by the header field value represented as a literal string as described in [Section 4.1.2](#).

[4.3.2](#). Literal Header Field with Incremental Indexing

A literal header field with incremental indexing adds a new entry to the header table.

Internet-Draft

HPACK

December 2013

```

    0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 |           Index (6+)       |
+---+---+---+---+---+---+---+---+
|           Value Length (8+)       |
+---+---+---+---+---+---+---+---+
| Value String (Length octets)     |
+---+---+---+---+---+---+---+---+

```

Literal Header Field with Incremental Indexing -
Indexed Name

```

    0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 |           0                 |
+---+---+---+---+---+---+---+---+
|           Name Length (8+)       |
+---+---+---+---+---+---+---+---+
| Name String (Length octets)     |
+---+---+---+---+---+---+---+---+
|           Value Length (8+)       |
+---+---+---+---+---+---+---+---+
| Value String (Length octets)     |
+---+---+---+---+---+---+---+---+

```

Literal Header Field with Incremental Indexing -
New Name

This representation starts with the '00' 2-bit pattern.

If the header field name matches the header field name of a (name, value) pair stored in the Header Table or Static Table, the header field name can be represented using the index of that entry. In this case, the index of the entry, index (which is strictly greater than 0), is represented as an integer with a 6-bit prefix (see [Section 4.1.1](#)).

Otherwise, the header field name is represented as a literal. The value 0 is represented on 6 bits followed by the header field name

(see [Section 4.1.2](#)).

The header field name representation is followed by the header field value represented as a literal string as described in [Section 4.1.2](#).

[5.](#) Security Considerations

This compressor exists to solve security issues present in stream compressors such as DEFLATE whereby the compression context can be efficiently probed to reveal secrets. A conformant implementation of this specification should be fairly safe against that kind of attack, as the reaping of any information from the compression context requires more work than guessing and verifying the plain text data directly with the server. As with any secret, however, the longer the length of the secret, the more difficult the secret is to guess. It is inadvisable to have short cookies that are relied upon to remain secret for any duration of time.

A proper security-conscious implementation will also need to prevent timing attacks by ensuring that the amount of time it takes to do string comparisons is always a function of the total length of the strings, and not a function of the number of matched characters.

A decoder needs to ensure that larger values or encodings of integers do not permit exploitation. Decoders MUST limit the size of integers, both in value and encoded length, that it accepts (see [Section 4.1.1](#)).

Another common security problem is when the remote endpoint successfully causes the local endpoint to exhaust its memory. This compressor attempts to deal with the most obvious ways that this could occur by limiting both the peak and the steady-state amount of memory consumed in the compressor state, by providing ways for the application to consume/flush the emitted header fields in small chunks, and by considering overhead in the state size calculation. Implementors must still be careful in the creation of APIs to an implementation of this compressor by ensuring that header field keys and values are either emitted as a stream, or that the compression

implementation have a limit on the maximum size of a key or value. Failure to implement these kinds of safeguards may still result in a scenario where the local endpoint exhausts its memory.

6. References

6.1. Normative References

[HTTP-p1] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [draft-ietf-httpbis-p1-messaging-25](#) (work in progress), November 2013.

[HTTP2] Belshe, M., Peon, R., Thomson, M., Ed., and A. Melnikov, Ed., "Hypertext Transfer Protocol version 2.0",

Peon & Ruellan

Expires June 7, 2014

[Page 18]

Internet-Draft

HPACK

December 2013

[draft-ietf-httpbis-http2-08](#) (work in progress),
November 2013.

6.2. Informative References

[CANON] Schwartz, E. and B. Kallick, "Generating a canonical prefix encoding", Communications of the ACM Volume 7 Issue 3, pp. 166-169, March 1964, <<http://dl.acm.org/citation.cfm?id=363991>>.

[CRIME] Rizzo, J. and T. Duong, "The Crime Attack", September 2012, <https://docs.google.com/a/twist.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2Gizeu0faLU2HOU/edit#slide=id.g1eb6c1b5_3_6>.

[HUFF] Huffman, D., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers Volume 40, Number 9, pp. 1098-1101, September 1952, <<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4051119>>.

[PERF1] Belshe, M., "IETF83: SPDY and What to Consider for HTTP/2.0", March 2012, <<http://www.ietf.org/proceedings/83/slides/slides-83-httpbis-3>>.

[PERF2] McManus, P., "SPDY: What I Like About You", September 2011, <<http://bitsup.blogspot.com/2011/09/spdy-what-i-like-about-you.html>>.

[SPDY] Belshe, M. and R. Peon, "SPDY Protocol", [draft-mbelshe-httpbis-spy-00](#) (work in progress), February 2012.

URIs

[1] <http://en.wikipedia.org/wiki/Variable-length_quantity>

[Appendix A](#). Change Log (to be removed by RFC Editor before publication)

[A.1](#). Since [draft-ietf-httpbis-header-compression-04](#)

- o Updated examples: take into account changes in the spec, and show more features.
- o Use 'octet' everywhere instead of having both 'byte' and 'octet'.

Peon & Ruellan

Expires June 7, 2014

[Page 19]

Internet-Draft

HPACK

December 2013

- o Added reference set emptying.
- o Editorial changes and clarifications.
- o Added "host" header to the static table.
- o Ordering for list of values (either NULL- or comma-separated).

[A.2](#). Since [draft-ietf-httpbis-header-compression-03](#)

- o A large number of editorial changes; changed the description of evicting/adding new entries.
- o Removed substitution indexing
- o Changed 'initial headers' to 'static headers', as per issue #258
- o Merged 'request' and 'response' static headers, as per issue #259

- o Changed text to indicate that new headers are added at index 0 and expire from the largest index, as per issue #233

[A.3.](#) Since [draft-ietf-httpbis-header-compression-02](#)

- o Corrected error in integer encoding pseudocode.

[A.4.](#) Since [draft-ietf-httpbis-header-compression-01](#)

- o Refactored of Header Encoding Section: split definitions and processing rule.
- o Backward incompatible change: Updated reference set management as per issue #214. This changes how the interaction between the reference set and eviction works. This also changes the working of the reference set in some specific cases.
- o Backward incompatible change: modified initial header list, as per issue #188.
- o Added example of 32 octets entry structure (issue #191).
- o Added Header Set Completion section. Reflowed some text. Clarified some writing which was awkward. Added text about duplicate header entry encoding. Clarified some language w.r.t Header Set. Changed x-my-header to mynewheader. Added text in the HeaderEmission section indicating that the application may also be able to free up memory more quickly. Added information in Security Considerations section.

[A.5.](#) Since [draft-ietf-httpbis-header-compression-01](#)

Fixed bug/omission in integer representation algorithm.

Changed the document title.

Header matching text rewritten.

Changed the definition of header emission.

Changed the name of the setting which dictates how much memory the compression context should use.

Removed "specific use cases" section

Corrected erroneous statement about what index can be contained in one octet

Added descriptions of opcodes

Removed security claims from introduction.

[Appendix B](#). Static Table

The static table consists of an unchangeable ordered list of (name, value) pairs. The first entry in the table is always represented by the index $\text{len}(\text{header table})+1$, and the last entry in the table is represented by the index $\text{len}(\text{header table})+\text{len}(\text{static table})$.

[[anchor9: The ordering of these tables is currently arbitrary. The tables in this section should be updated and ordered such that the table entries with the smallest indices are those which, based on a statistical analysis of the frequency of use weighted by size, achieve the largest decrease in octets transmitted subject to HTTP 2.0 header field rules (like removal of some header fields). This set of header fields is currently very likely incomplete, and should be made complete.]]

The following table lists the pre-defined header fields that make-up the static header table.

Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/

5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	500

10	:status	404
11	:status	403
12	:status	400
13	:status	401
14	accept-charset	
15	accept-encoding	
16	accept-language	
17	accept-ranges	
18	accept	
19	access-control-allow-origin	
20	age	
21	allow	
22	authorization	
23	cache-control	
24	content-disposition	
25	content-encoding	
26	content-language	
27	content-length	
28	content-location	
29	content-range	
30	content-type	
31	cookie	
32	date	
33	etag	
34	expect	
35	expires	
36	from	
37	host	
38	if-match	
39	if-modified-since	
40	if-none-match	
41	if-range	
42	if-unmodified-since	
43	last-modified	
44	link	
45	location	
46	max-forwards	
47	proxy-authenticate	
48	proxy-authorization	
49	range	
50	referer	
51	refresh	
52	retry-after	

53	server		
54	set-cookie		
55	strict-transport-security		
56	transfer-encoding		
57	user-agent		
58	vary		
59	via		
60	www-authenticate		

Table 1: Static Table Entries

The table give the index of each entry in the static table. The full index of each entry, to be used for encoding a reference to this entry, is computed by adding the number of entries in the header table to this index.

[Appendix C](#). Huffman Codes For Requests

The following Huffman codes are used when encoding string literals in the client to server direction.

[[anchor10: This table may need to be regenerated.]]

sym	aligned to MSB as bits	len in bits	aligned to LSB as hex	len in bits
(0)	11111111 11111111 11110111 010	[27]	7ffffba	[27]
(1)	11111111 11111111 11110111 011	[27]	7ffffbb	[27]
(2)	11111111 11111111 11110111 100	[27]	7ffffbc	[27]
(3)	11111111 11111111 11110111 101	[27]	7ffffbd	[27]
(4)	11111111 11111111 11110111 110	[27]	7ffffbe	[27]
(5)	11111111 11111111 11110111 111	[27]	7ffffbf	[27]
(6)	11111111 11111111 11111000 000	[27]	7ffffc0	[27]
(7)	11111111 11111111 11111000 001	[27]	7ffffc1	[27]
(8)	11111111 11111111 11111000 010	[27]	7ffffc2	[27]
(9)	11111111 11111111 11111000 011	[27]	7ffffc3	[27]
(10)	11111111 11111111 11111000 100	[27]	7ffffc4	[27]
(11)	11111111 11111111 11111000 101	[27]	7ffffc5	[27]
(12)	11111111 11111111 11111000 110	[27]	7ffffc6	[27]
(13)	11111111 11111111 11111000 111	[27]	7ffffc7	[27]
(14)	11111111 11111111 11111001 000	[27]	7ffffc8	[27]
(15)	11111111 11111111 11111001 001	[27]	7ffffc9	[27]
(16)	11111111 11111111 11111001 010	[27]	7ffffca	[27]
(17)	11111111 11111111 11111001 011	[27]	7ffffcb	[27]
(18)	11111111 11111111 11111001 100	[27]	7ffffcc	[27]
(19)	11111111 11111111 11111001 101	[27]	7ffffcd	[27]

Internet-Draft

HPACK

December 2013

(20)	11111111 11111111 11111001 110	[27]	7ffffce	[27]
(21)	11111111 11111111 11111001 111	[27]	7ffffcf	[27]
(22)	11111111 11111111 11111010 000	[27]	7ffffd0	[27]
(23)	11111111 11111111 11111010 001	[27]	7ffffd1	[27]
(24)	11111111 11111111 11111010 010	[27]	7ffffd2	[27]
(25)	11111111 11111111 11111010 011	[27]	7ffffd3	[27]
(26)	11111111 11111111 11111010 100	[27]	7ffffd4	[27]
(27)	11111111 11111111 11111010 101	[27]	7ffffd5	[27]
(28)	11111111 11111111 11111010 110	[27]	7ffffd6	[27]
(29)	11111111 11111111 11111010 111	[27]	7ffffd7	[27]
(30)	11111111 11111111 11111011 000	[27]	7ffffd8	[27]
(31)	11111111 11111111 11111011 001	[27]	7ffffd9	[27]
' '	(32) 11101000	[8]	e8	[8]
'!'	(33) 11111111 1100	[12]	ffc	[12]
'"'	(34) 11111111 111010	[14]	3ffa	[14]
'#'	(35) 11111111 1111100	[15]	7ffc	[15]
'\$'	(36) 11111111 1111101	[15]	7ffd	[15]
'%'	(37) 100100	[6]	24	[6]
'&'	(38) 1101110	[7]	6e	[7]
''''	(39) 11111111 1111110	[15]	7ffe	[15]
'('	(40) 11111111 010	[11]	7fa	[11]
')'	(41) 11111111 011	[11]	7fb	[11]
'*'	(42) 11111110 10	[10]	3fa	[10]
'+'	(43) 11111111 100	[11]	7fc	[11]
','	(44) 11101001	[8]	e9	[8]
'-'	(45) 100101	[6]	25	[6]
'.'	(46) 00100	[5]	4	[5]
'/'	(47) 0000	[4]	0	[4]
'0'	(48) 00101	[5]	5	[5]
'1'	(49) 00110	[5]	6	[5]
'2'	(50) 00111	[5]	7	[5]
'3'	(51) 100110	[6]	26	[6]
'4'	(52) 100111	[6]	27	[6]
'5'	(53) 101000	[6]	28	[6]
'6'	(54) 101001	[6]	29	[6]
'7'	(55) 101010	[6]	2a	[6]
'8'	(56) 101011	[6]	2b	[6]
'9'	(57) 101100	[6]	2c	[6]
':'	(58) 11110110 0	[9]	1ec	[9]
';'	(59) 11101010	[8]	ea	[8]
'<'	(60) 11111111 11111111 10	[18]	3ffe	[18]
'='	(61) 101101	[6]	2d	[6]

'>' (62)	11111111 11111110 0 [17]	1fffc [17]
'?' (63)	11110110 1 [9]	1ed [9]
'@' (64)	11111111 111011 [14]	3ffb [14]
'A' (65)	1101111 [7]	6f [7]
'B' (66)	11101011 [8]	eb [8]
'C' (67)	11101100 [8]	ec [8]

Internet-Draft

HPACK

December 2013

'D' (68)	11101101 [8]	ed [8]
'E' (69)	11101110 [8]	ee [8]
'F' (70)	1110000 [7]	70 [7]
'G' (71)	11110111 0 [9]	1ee [9]
'H' (72)	11110111 1 [9]	1ef [9]
'I' (73)	11111000 0 [9]	1f0 [9]
'J' (74)	11111000 1 [9]	1f1 [9]
'K' (75)	11111110 11 [10]	3fb [10]
'L' (76)	11111001 0 [9]	1f2 [9]
'M' (77)	11101111 [8]	ef [8]
'N' (78)	11111001 1 [9]	1f3 [9]
'O' (79)	11111010 0 [9]	1f4 [9]
'P' (80)	11111010 1 [9]	1f5 [9]
'Q' (81)	11111011 0 [9]	1f6 [9]
'R' (82)	11111011 1 [9]	1f7 [9]
'S' (83)	11110000 [8]	f0 [8]
'T' (84)	11110001 [8]	f1 [8]
'U' (85)	11111100 0 [9]	1f8 [9]
'V' (86)	11111100 1 [9]	1f9 [9]
'W' (87)	11111101 0 [9]	1fa [9]
'X' (88)	11111101 1 [9]	1fb [9]
'Y' (89)	11111110 0 [9]	1fc [9]
'Z' (90)	11111111 00 [10]	3fc [10]
'[' (91)	11111111 111100 [14]	3ffc [14]
'\' (92)	11111111 11111111 11111011 010 [27]	7ffffda [27]
']' (93)	11111111 11100 [13]	1ffc [13]
'^' (94)	11111111 111101 [14]	3ffd [14]
'_' (95)	101110 [6]	2e [6]
'`' (96)	11111111 11111111 110 [19]	7fffe [19]
'a' (97)	01000 [5]	8 [5]
'b' (98)	101111 [6]	2f [6]
'c' (99)	01001 [5]	9 [5]
'd' (100)	110000 [6]	30 [6]
'e' (101)	0001 [4]	1 [4]
'f' (102)	110001 [6]	31 [6]

'g'	(103)	110010	[6]	32	[6]
'h'	(104)	110011	[6]	33	[6]
'i'	(105)	01010	[5]	a	[5]
'j'	(106)	1110001	[7]	71	[7]
'k'	(107)	1110010	[7]	72	[7]
'l'	(108)	01011	[5]	b	[5]
'm'	(109)	110100	[6]	34	[6]
'n'	(110)	01100	[5]	c	[5]
'o'	(111)	01101	[5]	d	[5]
'p'	(112)	01110	[5]	e	[5]
'q'	(113)	11110010	[8]	f2	[8]
'r'	(114)	01111	[5]	f	[5]
's'	(115)	10000	[5]	10	[5]

't'	(116)	10001	[5]	11	[5]
'u'	(117)	110101	[6]	35	[6]
'v'	(118)	1110011	[7]	73	[7]
'w'	(119)	110110	[6]	36	[6]
'x'	(120)	11110011	[8]	f3	[8]
'y'	(121)	11110100	[8]	f4	[8]
'z'	(122)	11110101	[8]	f5	[8]
'{'	(123)	11111111 11111110 1	[17]	1fffd	[17]
' '	(124)	11111111 101	[11]	7fd	[11]
'}'	(125)	11111111 11111111 0	[17]	1fffe	[17]
'~'	(126)	11111111 1101	[12]	ffd	[12]
	(127)	11111111 11111111 11111011 011	[27]	7ffffdb	[27]
	(128)	11111111 11111111 11111011 100	[27]	7ffffdc	[27]
	(129)	11111111 11111111 11111011 101	[27]	7ffffdd	[27]
	(130)	11111111 11111111 11111011 110	[27]	7ffffde	[27]
	(131)	11111111 11111111 11111011 111	[27]	7ffffdf	[27]
	(132)	11111111 11111111 11111100 000	[27]	7ffffe0	[27]
	(133)	11111111 11111111 11111100 001	[27]	7ffffe1	[27]
	(134)	11111111 11111111 11111100 010	[27]	7ffffe2	[27]
	(135)	11111111 11111111 11111100 011	[27]	7ffffe3	[27]
	(136)	11111111 11111111 11111100 100	[27]	7ffffe4	[27]
	(137)	11111111 11111111 11111100 101	[27]	7ffffe5	[27]
	(138)	11111111 11111111 11111100 110	[27]	7ffffe6	[27]
	(139)	11111111 11111111 11111100 111	[27]	7ffffe7	[27]
	(140)	11111111 11111111 11111101 000	[27]	7ffffe8	[27]
	(141)	11111111 11111111 11111101 001	[27]	7ffffe9	[27]
	(142)	11111111 11111111 11111101 010	[27]	7ffffea	[27]
	(143)	11111111 11111111 11111101 011	[27]	7ffffeb	[27]

(144)		11111111		11111111		11111101		100	[27]	7ffffec	[27]
(145)		11111111		11111111		11111101		101	[27]	7ffffed	[27]
(146)		11111111		11111111		11111101		110	[27]	7ffffee	[27]
(147)		11111111		11111111		11111101		111	[27]	7ffffef	[27]
(148)		11111111		11111111		11111110		000	[27]	7fffff0	[27]
(149)		11111111		11111111		11111110		001	[27]	7fffff1	[27]
(150)		11111111		11111111		11111110		010	[27]	7fffff2	[27]
(151)		11111111		11111111		11111110		011	[27]	7fffff3	[27]
(152)		11111111		11111111		11111110		100	[27]	7fffff4	[27]
(153)		11111111		11111111		11111110		101	[27]	7fffff5	[27]
(154)		11111111		11111111		11111110		110	[27]	7fffff6	[27]
(155)		11111111		11111111		11111110		111	[27]	7fffff7	[27]
(156)		11111111		11111111		11111111		000	[27]	7fffff8	[27]
(157)		11111111		11111111		11111111		001	[27]	7fffff9	[27]
(158)		11111111		11111111		11111111		010	[27]	7fffffa	[27]
(159)		11111111		11111111		11111111		011	[27]	7fffffb	[27]
(160)		11111111		11111111		11111111		100	[27]	7fffffc	[27]
(161)		11111111		11111111		11111111		101	[27]	7fffffd	[27]
(162)		11111111		11111111		11111111		110	[27]	7fffffe	[27]
(163)		11111111		11111111		11111111		111	[27]	7ffffff	[27]

(164)		11111111		11111111		11100000		00	[26]	3ffff80	[26]
(165)		11111111		11111111		11100000		01	[26]	3ffff81	[26]
(166)		11111111		11111111		11100000		10	[26]	3ffff82	[26]
(167)		11111111		11111111		11100000		11	[26]	3ffff83	[26]
(168)		11111111		11111111		11100001		00	[26]	3ffff84	[26]
(169)		11111111		11111111		11100001		01	[26]	3ffff85	[26]
(170)		11111111		11111111		11100001		10	[26]	3ffff86	[26]
(171)		11111111		11111111		11100001		11	[26]	3ffff87	[26]
(172)		11111111		11111111		11100010		00	[26]	3ffff88	[26]
(173)		11111111		11111111		11100010		01	[26]	3ffff89	[26]
(174)		11111111		11111111		11100010		10	[26]	3ffff8a	[26]
(175)		11111111		11111111		11100010		11	[26]	3ffff8b	[26]
(176)		11111111		11111111		11100011		00	[26]	3ffff8c	[26]
(177)		11111111		11111111		11100011		01	[26]	3ffff8d	[26]
(178)		11111111		11111111		11100011		10	[26]	3ffff8e	[26]
(179)		11111111		11111111		11100011		11	[26]	3ffff8f	[26]
(180)		11111111		11111111		11100100		00	[26]	3ffff90	[26]
(181)		11111111		11111111		11100100		01	[26]	3ffff91	[26]
(182)		11111111		11111111		11100100		10	[26]	3ffff92	[26]
(183)		11111111		11111111		11100100		11	[26]	3ffff93	[26]
(184)		11111111		11111111		11100101		00	[26]	3ffff94	[26]

(185)		11111111		11111111		11100101		01	[26]	3ffff95	[26]
(186)		11111111		11111111		11100101		10	[26]	3ffff96	[26]
(187)		11111111		11111111		11100101		11	[26]	3ffff97	[26]
(188)		11111111		11111111		11100110		00	[26]	3ffff98	[26]
(189)		11111111		11111111		11100110		01	[26]	3ffff99	[26]
(190)		11111111		11111111		11100110		10	[26]	3ffff9a	[26]
(191)		11111111		11111111		11100110		11	[26]	3ffff9b	[26]
(192)		11111111		11111111		11100111		00	[26]	3ffff9c	[26]
(193)		11111111		11111111		11100111		01	[26]	3ffff9d	[26]
(194)		11111111		11111111		11100111		10	[26]	3ffff9e	[26]
(195)		11111111		11111111		11100111		11	[26]	3ffff9f	[26]
(196)		11111111		11111111		11101000		00	[26]	3ffffa0	[26]
(197)		11111111		11111111		11101000		01	[26]	3ffffa1	[26]
(198)		11111111		11111111		11101000		10	[26]	3ffffa2	[26]
(199)		11111111		11111111		11101000		11	[26]	3ffffa3	[26]
(200)		11111111		11111111		11101001		00	[26]	3ffffa4	[26]
(201)		11111111		11111111		11101001		01	[26]	3ffffa5	[26]
(202)		11111111		11111111		11101001		10	[26]	3ffffa6	[26]
(203)		11111111		11111111		11101001		11	[26]	3ffffa7	[26]
(204)		11111111		11111111		11101010		00	[26]	3ffffa8	[26]
(205)		11111111		11111111		11101010		01	[26]	3ffffa9	[26]
(206)		11111111		11111111		11101010		10	[26]	3ffffaa	[26]
(207)		11111111		11111111		11101010		11	[26]	3ffffab	[26]
(208)		11111111		11111111		11101011		00	[26]	3ffffac	[26]
(209)		11111111		11111111		11101011		01	[26]	3ffffad	[26]
(210)		11111111		11111111		11101011		10	[26]	3ffffae	[26]
(211)		11111111		11111111		11101011		11	[26]	3ffffaf	[26]

(212)		11111111		11111111		11101100		00	[26]	3ffffb0	[26]
(213)		11111111		11111111		11101100		01	[26]	3ffffb1	[26]
(214)		11111111		11111111		11101100		10	[26]	3ffffb2	[26]
(215)		11111111		11111111		11101100		11	[26]	3ffffb3	[26]
(216)		11111111		11111111		11101101		00	[26]	3ffffb4	[26]
(217)		11111111		11111111		11101101		01	[26]	3ffffb5	[26]
(218)		11111111		11111111		11101101		10	[26]	3ffffb6	[26]
(219)		11111111		11111111		11101101		11	[26]	3ffffb7	[26]
(220)		11111111		11111111		11101110		00	[26]	3ffffb8	[26]
(221)		11111111		11111111		11101110		01	[26]	3ffffb9	[26]
(222)		11111111		11111111		11101110		10	[26]	3ffffba	[26]
(223)		11111111		11111111		11101110		11	[26]	3ffffbb	[26]
(224)		11111111		11111111		11101111		00	[26]	3ffffbc	[26]
(225)		11111111		11111111		11101111		01	[26]	3ffffbd	[26]

(226)	11111111 11111111 11101111 10	[26]	3ffffbe	[26]
(227)	11111111 11111111 11101111 11	[26]	3ffffbf	[26]
(228)	11111111 11111111 11110000 00	[26]	3ffffc0	[26]
(229)	11111111 11111111 11110000 01	[26]	3ffffc1	[26]
(230)	11111111 11111111 11110000 10	[26]	3ffffc2	[26]
(231)	11111111 11111111 11110000 11	[26]	3ffffc3	[26]
(232)	11111111 11111111 11110001 00	[26]	3ffffc4	[26]
(233)	11111111 11111111 11110001 01	[26]	3ffffc5	[26]
(234)	11111111 11111111 11110001 10	[26]	3ffffc6	[26]
(235)	11111111 11111111 11110001 11	[26]	3ffffc7	[26]
(236)	11111111 11111111 11110010 00	[26]	3ffffc8	[26]
(237)	11111111 11111111 11110010 01	[26]	3ffffc9	[26]
(238)	11111111 11111111 11110010 10	[26]	3ffffca	[26]
(239)	11111111 11111111 11110010 11	[26]	3ffffcb	[26]
(240)	11111111 11111111 11110011 00	[26]	3ffffcc	[26]
(241)	11111111 11111111 11110011 01	[26]	3ffffcd	[26]
(242)	11111111 11111111 11110011 10	[26]	3ffffce	[26]
(243)	11111111 11111111 11110011 11	[26]	3ffffcf	[26]
(244)	11111111 11111111 11110100 00	[26]	3ffffd0	[26]
(245)	11111111 11111111 11110100 01	[26]	3ffffd1	[26]
(246)	11111111 11111111 11110100 10	[26]	3ffffd2	[26]
(247)	11111111 11111111 11110100 11	[26]	3ffffd3	[26]
(248)	11111111 11111111 11110101 00	[26]	3ffffd4	[26]
(249)	11111111 11111111 11110101 01	[26]	3ffffd5	[26]
(250)	11111111 11111111 11110101 10	[26]	3ffffd6	[26]
(251)	11111111 11111111 11110101 11	[26]	3ffffd7	[26]
(252)	11111111 11111111 11110110 00	[26]	3ffffd8	[26]
(253)	11111111 11111111 11110110 01	[26]	3ffffd9	[26]
(254)	11111111 11111111 11110110 10	[26]	3ffffda	[26]
(255)	11111111 11111111 11110110 11	[26]	3ffffdb	[26]
EOS (256)	11111111 11111111 11110111 00	[26]	3ffffdc	[26]

[Appendix D](#). Huffman Codes for Responses

The following Huffman codes are used when encoding string literals in the server to client direction. These codes apply for both responses to client requests and for push-promises.

[[anchor11: This table may need to be regenerated.]]

sym	aligned to MSB as bits				len in bits	aligned to LSB as hex		len in bits
(0)	11111111	11111111	11011110	0 [25]		1ffffbc	[25]	
(1)	11111111	11111111	11011110	1 [25]		1ffffbd	[25]	
(2)	11111111	11111111	11011111	0 [25]		1ffffbe	[25]	
(3)	11111111	11111111	11011111	1 [25]		1ffffbf	[25]	
(4)	11111111	11111111	11100000	0 [25]		1ffffc0	[25]	
(5)	11111111	11111111	11100000	1 [25]		1ffffc1	[25]	
(6)	11111111	11111111	11100001	0 [25]		1ffffc2	[25]	
(7)	11111111	11111111	11100001	1 [25]		1ffffc3	[25]	
(8)	11111111	11111111	11100010	0 [25]		1ffffc4	[25]	
(9)	11111111	11111111	11100010	1 [25]		1ffffc5	[25]	
(10)	11111111	11111111	11100011	0 [25]		1ffffc6	[25]	
(11)	11111111	11111111	11100011	1 [25]		1ffffc7	[25]	
(12)	11111111	11111111	11100100	0 [25]		1ffffc8	[25]	
(13)	11111111	11111111	11100100	1 [25]		1ffffc9	[25]	
(14)	11111111	11111111	11100101	0 [25]		1ffffca	[25]	
(15)	11111111	11111111	11100101	1 [25]		1ffffcb	[25]	
(16)	11111111	11111111	11100110	0 [25]		1ffffcc	[25]	
(17)	11111111	11111111	11100110	1 [25]		1ffffcd	[25]	
(18)	11111111	11111111	11100111	0 [25]		1ffffce	[25]	
(19)	11111111	11111111	11100111	1 [25]		1ffffcf	[25]	
(20)	11111111	11111111	11101000	0 [25]		1ffffd0	[25]	
(21)	11111111	11111111	11101000	1 [25]		1ffffd1	[25]	
(22)	11111111	11111111	11101001	0 [25]		1ffffd2	[25]	
(23)	11111111	11111111	11101001	1 [25]		1ffffd3	[25]	
(24)	11111111	11111111	11101010	0 [25]		1ffffd4	[25]	
(25)	11111111	11111111	11101010	1 [25]		1ffffd5	[25]	
(26)	11111111	11111111	11101011	0 [25]		1ffffd6	[25]	
(27)	11111111	11111111	11101011	1 [25]		1ffffd7	[25]	
(28)	11111111	11111111	11101100	0 [25]		1ffffd8	[25]	
(29)	11111111	11111111	11101100	1 [25]		1ffffd9	[25]	
(30)	11111111	11111111	11101101	0 [25]		1ffffda	[25]	
(31)	11111111	11111111	11101101	1 [25]		1ffffdb	[25]	
' ' (32)	0000 [4]						0 [4]	
'!' (33)	11111111	1010 [12]				ffa [12]		
'"' (34)	1101010 [7]					6a [7]		

'#' (35) 11111111 11010 [13]	1ffa [13]
'\$' (36) 11111111 111100 [14]	3ffc [14]
'%' (37) 11110110 0 [9]	1ec [9]
'&' (38) 11111110 00 [10]	3f8 [10]
''' (39) 11111111 11011 [13]	1ffb [13]
'(' (40) 11110110 1 [9]	1ed [9]
)' (41) 11110111 0 [9]	1ee [9]
'*' (42) 11111111 1011 [12]	ffb [12]
'+' (43) 11111111 010 [11]	7fa [11]
',' (44) 100010 [6]	22 [6]
'-' (45) 100011 [6]	23 [6]
'.' (46) 100100 [6]	24 [6]
'/' (47) 1101011 [7]	6b [7]
'0' (48) 0001 [4]	1 [4]
'1' (49) 0010 [4]	2 [4]
'2' (50) 0011 [4]	3 [4]
'3' (51) 01000 [5]	8 [5]
'4' (52) 01001 [5]	9 [5]
'5' (53) 01010 [5]	a [5]
'6' (54) 100101 [6]	25 [6]
'7' (55) 100110 [6]	26 [6]
'8' (56) 01011 [5]	b [5]
'9' (57) 01100 [5]	c [5]
':' (58) 01101 [5]	d [5]
';' (59) 11110111 1 [9]	1ef [9]
'<' (60) 11111111 11111010 [16]	fffa [16]
'=' (61) 1101100 [7]	6c [7]
'>' (62) 11111111 11100 [13]	1ffc [13]
'?' (63) 11111111 1100 [12]	ffc [12]
'@' (64) 11111111 11111011 [16]	fffb [16]
'A' (65) 1101101 [7]	6d [7]
'B' (66) 11101010 [8]	ea [8]
'C' (67) 11101011 [8]	eb [8]
'D' (68) 11101100 [8]	ec [8]
'E' (69) 11101101 [8]	ed [8]
'F' (70) 11101110 [8]	ee [8]
'G' (71) 100111 [6]	27 [6]
'H' (72) 11111000 0 [9]	1f0 [9]
'I' (73) 11101111 [8]	ef [8]
'J' (74) 11110000 [8]	f0 [8]
'K' (75) 11111110 01 [10]	3f9 [10]
'L' (76) 11111000 1 [9]	1f1 [9]
'M' (77) 101000 [6]	28 [6]
'N' (78) 11110001 [8]	f1 [8]
'O' (79) 11110010 [8]	f2 [8]
'P' (80) 11111001 0 [9]	1f2 [9]
'Q' (81) 11111110 10 [10]	3fa [10]
'R' (82) 11111001 1 [9]	1f3 [9]

Internet-Draft

HPACK

December 2013

'S' (83)	101001 [6]	29 [6]
'T' (84)	01110 [5]	e [5]
'U' (85)	11111010 0 [9]	1f4 [9]
'V' (86)	11111010 1 [9]	1f5 [9]
'W' (87)	11110011 [8]	f3 [8]
'X' (88)	11111110 11 [10]	3fb [10]
'Y' (89)	11111011 0 [9]	1f6 [9]
'Z' (90)	11111111 00 [10]	3fc [10]
'[' (91)	11111111 011 [11]	7fb [11]
'\' (92)	11111111 11101 [13]	1ffd [13]
']' (93)	11111111 100 [11]	7fc [11]
'^' (94)	11111111 1111100 [15]	7ffc [15]
'_' (95)	11111011 1 [9]	1f7 [9]
'`' (96)	11111111 11111111 0 [17]	1fffe [17]
'a' (97)	01111 [5]	f [5]
'b' (98)	1101110 [7]	6e [7]
'c' (99)	101010 [6]	2a [6]
'd' (100)	101011 [6]	2b [6]
'e' (101)	10000 [5]	10 [5]
'f' (102)	1101111 [7]	6f [7]
'g' (103)	1110000 [7]	70 [7]
'h' (104)	1110001 [7]	71 [7]
'i' (105)	101100 [6]	2c [6]
'j' (106)	11111100 0 [9]	1f8 [9]
'k' (107)	11111100 1 [9]	1f9 [9]
'l' (108)	1110010 [7]	72 [7]
'm' (109)	101101 [6]	2d [6]
'n' (110)	101110 [6]	2e [6]
'o' (111)	101111 [6]	2f [6]
'p' (112)	110000 [6]	30 [6]
'q' (113)	11111101 0 [9]	1fa [9]
'r' (114)	110001 [6]	31 [6]
's' (115)	110010 [6]	32 [6]
't' (116)	110011 [6]	33 [6]
'u' (117)	110100 [6]	34 [6]
'v' (118)	1110011 [7]	73 [7]
'w' (119)	11110100 [8]	f4 [8]
'x' (120)	1110100 [7]	74 [7]
'y' (121)	11110101 [8]	f5 [8]
'z' (122)	11111101 1 [9]	1fb [9]
'{' (123)	11111111 11111100 [16]	fffc [16]
' ' (124)	11111111 111101 [14]	3ffd [14]
'}' (125)	11111111 11111101 [16]	fffd [16]

'~'	(126)	11111111 11111110	[16]		fffe	[16]
	(127)	11111111 11111111 11101110	0 [25]		1ffffdc	[25]
	(128)	11111111 11111111 11101110	1 [25]		1ffffdd	[25]
	(129)	11111111 11111111 11101111	0 [25]		1ffffde	[25]
	(130)	11111111 11111111 11101111	1 [25]		1ffffdf	[25]

(131)	11111111 11111111 11110000	0 [25]	1ffffe0	[25]
(132)	11111111 11111111 11110000	1 [25]	1ffffe1	[25]
(133)	11111111 11111111 11110001	0 [25]	1ffffe2	[25]
(134)	11111111 11111111 11110001	1 [25]	1ffffe3	[25]
(135)	11111111 11111111 11110010	0 [25]	1ffffe4	[25]
(136)	11111111 11111111 11110010	1 [25]	1ffffe5	[25]
(137)	11111111 11111111 11110011	0 [25]	1ffffe6	[25]
(138)	11111111 11111111 11110011	1 [25]	1ffffe7	[25]
(139)	11111111 11111111 11110100	0 [25]	1ffffe8	[25]
(140)	11111111 11111111 11110100	1 [25]	1ffffe9	[25]
(141)	11111111 11111111 11110101	0 [25]	1ffffea	[25]
(142)	11111111 11111111 11110101	1 [25]	1ffffeb	[25]
(143)	11111111 11111111 11110110	0 [25]	1ffffec	[25]
(144)	11111111 11111111 11110110	1 [25]	1ffffed	[25]
(145)	11111111 11111111 11110111	0 [25]	1ffffee	[25]
(146)	11111111 11111111 11110111	1 [25]	1ffffef	[25]
(147)	11111111 11111111 11111000	0 [25]	1fffff0	[25]
(148)	11111111 11111111 11111000	1 [25]	1fffff1	[25]
(149)	11111111 11111111 11111001	0 [25]	1fffff2	[25]
(150)	11111111 11111111 11111001	1 [25]	1fffff3	[25]
(151)	11111111 11111111 11111010	0 [25]	1fffff4	[25]
(152)	11111111 11111111 11111010	1 [25]	1fffff5	[25]
(153)	11111111 11111111 11111011	0 [25]	1fffff6	[25]
(154)	11111111 11111111 11111011	1 [25]	1fffff7	[25]
(155)	11111111 11111111 11111100	0 [25]	1fffff8	[25]
(156)	11111111 11111111 11111100	1 [25]	1fffff9	[25]
(157)	11111111 11111111 11111101	0 [25]	1fffffa	[25]
(158)	11111111 11111111 11111101	1 [25]	1fffffb	[25]
(159)	11111111 11111111 11111110	0 [25]	1fffffc	[25]
(160)	11111111 11111111 11111110	1 [25]	1fffffd	[25]
(161)	11111111 11111111 11111111	0 [25]	1fffffe	[25]
(162)	11111111 11111111 11111111	1 [25]	1ffffff	[25]
(163)	11111111 11111111 10000000	[24]	ffff80	[24]
(164)	11111111 11111111 10000001	[24]	ffff81	[24]
(165)	11111111 11111111 10000010	[24]	ffff82	[24]
(166)	11111111 11111111 10000011	[24]	ffff83	[24]

(167)	11111111	11111111	10000100	[24]	ffff84	[24]
(168)	11111111	11111111	10000101	[24]	ffff85	[24]
(169)	11111111	11111111	10000110	[24]	ffff86	[24]
(170)	11111111	11111111	10000111	[24]	ffff87	[24]
(171)	11111111	11111111	10001000	[24]	ffff88	[24]
(172)	11111111	11111111	10001001	[24]	ffff89	[24]
(173)	11111111	11111111	10001010	[24]	ffff8a	[24]
(174)	11111111	11111111	10001011	[24]	ffff8b	[24]
(175)	11111111	11111111	10001100	[24]	ffff8c	[24]
(176)	11111111	11111111	10001101	[24]	ffff8d	[24]
(177)	11111111	11111111	10001110	[24]	ffff8e	[24]
(178)	11111111	11111111	10001111	[24]	ffff8f	[24]

(179)	11111111	11111111	10010000	[24]	ffff90	[24]
(180)	11111111	11111111	10010001	[24]	ffff91	[24]
(181)	11111111	11111111	10010010	[24]	ffff92	[24]
(182)	11111111	11111111	10010011	[24]	ffff93	[24]
(183)	11111111	11111111	10010100	[24]	ffff94	[24]
(184)	11111111	11111111	10010101	[24]	ffff95	[24]
(185)	11111111	11111111	10010110	[24]	ffff96	[24]
(186)	11111111	11111111	10010111	[24]	ffff97	[24]
(187)	11111111	11111111	10011000	[24]	ffff98	[24]
(188)	11111111	11111111	10011001	[24]	ffff99	[24]
(189)	11111111	11111111	10011010	[24]	ffff9a	[24]
(190)	11111111	11111111	10011011	[24]	ffff9b	[24]
(191)	11111111	11111111	10011100	[24]	ffff9c	[24]
(192)	11111111	11111111	10011101	[24]	ffff9d	[24]
(193)	11111111	11111111	10011110	[24]	ffff9e	[24]
(194)	11111111	11111111	10011111	[24]	ffff9f	[24]
(195)	11111111	11111111	10100000	[24]	ffffa0	[24]
(196)	11111111	11111111	10100001	[24]	ffffa1	[24]
(197)	11111111	11111111	10100010	[24]	ffffa2	[24]
(198)	11111111	11111111	10100011	[24]	ffffa3	[24]
(199)	11111111	11111111	10100100	[24]	ffffa4	[24]
(200)	11111111	11111111	10100101	[24]	ffffa5	[24]
(201)	11111111	11111111	10100110	[24]	ffffa6	[24]
(202)	11111111	11111111	10100111	[24]	ffffa7	[24]
(203)	11111111	11111111	10101000	[24]	ffffa8	[24]
(204)	11111111	11111111	10101001	[24]	ffffa9	[24]
(205)	11111111	11111111	10101010	[24]	ffffaa	[24]
(206)	11111111	11111111	10101011	[24]	ffffab	[24]
(207)	11111111	11111111	10101100	[24]	ffffac	[24]

(208)	11111111	11111111	10101101	[24]	ffffad	[24]
(209)	11111111	11111111	10101110	[24]	ffffae	[24]
(210)	11111111	11111111	10101111	[24]	ffffaf	[24]
(211)	11111111	11111111	10110000	[24]	ffffb0	[24]
(212)	11111111	11111111	10110001	[24]	ffffb1	[24]
(213)	11111111	11111111	10110010	[24]	ffffb2	[24]
(214)	11111111	11111111	10110011	[24]	ffffb3	[24]
(215)	11111111	11111111	10110100	[24]	ffffb4	[24]
(216)	11111111	11111111	10110101	[24]	ffffb5	[24]
(217)	11111111	11111111	10110110	[24]	ffffb6	[24]
(218)	11111111	11111111	10110111	[24]	ffffb7	[24]
(219)	11111111	11111111	10111000	[24]	ffffb8	[24]
(220)	11111111	11111111	10111001	[24]	ffffb9	[24]
(221)	11111111	11111111	10111010	[24]	ffffba	[24]
(222)	11111111	11111111	10111011	[24]	ffffbb	[24]
(223)	11111111	11111111	10111100	[24]	ffffbc	[24]
(224)	11111111	11111111	10111101	[24]	ffffbd	[24]
(225)	11111111	11111111	10111110	[24]	ffffbe	[24]
(226)	11111111	11111111	10111111	[24]	ffffbf	[24]

(227)	11111111	11111111	11000000	[24]	ffffc0	[24]
(228)	11111111	11111111	11000001	[24]	ffffc1	[24]
(229)	11111111	11111111	11000010	[24]	ffffc2	[24]
(230)	11111111	11111111	11000011	[24]	ffffc3	[24]
(231)	11111111	11111111	11000100	[24]	ffffc4	[24]
(232)	11111111	11111111	11000101	[24]	ffffc5	[24]
(233)	11111111	11111111	11000110	[24]	ffffc6	[24]
(234)	11111111	11111111	11000111	[24]	ffffc7	[24]
(235)	11111111	11111111	11001000	[24]	ffffc8	[24]
(236)	11111111	11111111	11001001	[24]	ffffc9	[24]
(237)	11111111	11111111	11001010	[24]	ffffca	[24]
(238)	11111111	11111111	11001011	[24]	ffffcb	[24]
(239)	11111111	11111111	11001100	[24]	ffffcc	[24]
(240)	11111111	11111111	11001101	[24]	ffffcd	[24]
(241)	11111111	11111111	11001110	[24]	ffffce	[24]
(242)	11111111	11111111	11001111	[24]	ffffcf	[24]
(243)	11111111	11111111	11010000	[24]	ffffd0	[24]
(244)	11111111	11111111	11010001	[24]	ffffd1	[24]
(245)	11111111	11111111	11010010	[24]	ffffd2	[24]
(246)	11111111	11111111	11010011	[24]	ffffd3	[24]
(247)	11111111	11111111	11010100	[24]	ffffd4	[24]
(248)	11111111	11111111	11010101	[24]	ffffd5	[24]

Header table (after decoding): empty.

Decoded header set:

:path: /sample/path

[E.1.3.](#) Indexed Header Field

The header field representation uses an indexed header field, from the static table. Upon using it, the static table entry is copied into the header table.

Header set to encode:

:method: GET

Reference set: empty.

Hex dump of encoded data:

```
82                                | .
```

Decoding process:

```
82                                | == Indexed - Add ==  
                                |   idx = 2  
                                | -> :method: GET
```

Header Table (after decoding):

```
[ 1] (s = 42) :method: GET  
      Table size: 42
```

Decoded header set:

:method: GET

[E.1.4.](#) Indexed Header Field from Static Table

The header field representation uses an indexed header field, from the static table. In this example, the `SETTINGS_HEADER_TABLE_SIZE` is set to 0, therefore, the entry is not copied into the header table.

Header set to encode:

```
:method: GET
```

Reference set: empty.

Hex dump of encoded data:

```
82                                | .
```

Decoding process:

```
82                                | == Indexed - Add ==  
                                |   idx = 2  
                                | -> :method: GET
```

Header table (after decoding): empty.

Decoded header set:

```
:method: GET
```

[E.2.](#) Request Examples without Huffman

This section shows several consecutive header sets, corresponding to HTTP requests, on the same connection.

[E.2.1.](#) First request

Header set to encode:

```
:method: GET  
:scheme: http  
:path: /  
:authority: www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
8287 8604 0f77 7777 2e65 7861 6d70 6c65 | .....www.example
2e63 6f6d | .com
```

Decoding process:

```
82 | == Indexed - Add ==
   |   idx = 2
   | -> :method: GET
87 | == Indexed - Add ==
   |   idx = 7
   | -> :scheme: http
86 | == Indexed - Add ==
   |   idx = 6
   | -> :path: /
04 | == Literal indexed ==
   |   Indexed name (idx = 4)
   |   :authority
0f |   Literal value (len = 15)
7777 772e 6578 616d 706c 652e 636f 6d | www.example.com
   | -> :authority: www.example\
   |   .com
```

Header Table (after decoding):

```
[ 1] (s = 57) :authority: www.example.com
[ 2] (s = 38) :path: /
[ 3] (s = 43) :scheme: http
[ 4] (s = 42) :method: GET
    Table size: 180
```

Decoded header set:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

[E.2.2.](#) Second request

This request takes advantage of the differential encoding of header sets.

Internet-Draft

HPACK

December 2013

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
cache-control: no-cache
```

Reference set:

```
[ 1] :authority: www.example.com
[ 2] :path: /
[ 3] :scheme: http
[ 4] :method: GET
```

Hex dump of encoded data:

```
1b08 6e6f 2d63 6163 6865      | ..no-cache
```

Decoding process:

```
1b      | == Literal indexed ==
         |   Indexed name (idx = 27)
         |   cache-control
08      |   Literal value (len = 8)
6e6f 2d63 6163 6865 | no-cache
         | -> cache-control: no-cache
```

Header Table (after decoding):

```
[ 1] (s = 53) cache-control: no-cache
[ 2] (s = 57) :authority: www.example.com
[ 3] (s = 38) :path: /
[ 4] (s = 43) :scheme: http
[ 5] (s = 42) :method: GET
      Table size: 233
```

Decoded header set:

```
cache-control: no-cache
:authority: www.example.com
:path: /
:scheme: http
:method: GET
```

[E.2.3.](#) Third request

This request has not enough headers in common with the previous request to take advantage of the differential encoding. Therefore, the reference set is emptied before encoding the header fields.

Header set to encode:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

Reference set:

```
[ 1] cache-control: no-cache
[ 2] :authority: www.example.com
[ 3] :path: /
[ 4] :scheme: http
[ 5] :method: GET
```

Hex dump of encoded data:

```
8085 8c8b 8400 0a63 7573 746f 6d2d 6b65 | .....custom-ke
790c 6375 7374 6f6d 2d76 616c 7565      | y.custom-value
```

Internet-Draft

HPACK

December 2013

Decoding process:

```
80      | == Empty reference set ==
      |   idx = 0
85      | == Indexed - Add ==
      |   idx = 5
      |   -> :method: GET
8c      | == Indexed - Add ==
      |   idx = 12
      |   -> :scheme: https
8b      | == Indexed - Add ==
      |   idx = 11
      |   -> :path: /index.html
84      | == Indexed - Add ==
      |   idx = 4
      |   -> :authority: www.example\
      |   .com
00      | == Literal indexed ==
0a      |   Literal name (len = 10)
6375 7374 6f6d 2d6b 6579 | custom-key
0c      |   Literal value (len = 12)
6375 7374 6f6d 2d76 616c 7565 | custom-value
      | -> custom-key: custom-valu\
      | e
```

Header Table (after decoding):

```
[ 1] (s = 54) custom-key: custom-value
[ 2] (s = 48) :path: /index.html
[ 3] (s = 44) :scheme: https
[ 4] (s = 53) cache-control: no-cache
[ 5] (s = 57) :authority: www.example.com
[ 6] (s = 38) :path: /
[ 7] (s = 43) :scheme: http
[ 8] (s = 42) :method: GET
    Table size: 379
```

Decoded header set:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

[E.3.](#) Request Examples with Huffman

This section shows the same examples as the previous section, but using Huffman encoding for the literal values.

[E.3.1.](#) First request

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

Reference set: empty.

Hex dump of encoded data:

8287 8604 8bdb 6d88 3e68 d1cb 1225 ba7f |m..h...%..

Decoding process:

```
82 | == Indexed - Add ==
   |   idx = 2
87 | -> :method: GET
   | == Indexed - Add ==
   |   idx = 7
86 | -> :scheme: http
   | == Indexed - Add ==
   |   idx = 6
04 | -> :path: /
   | == Literal indexed ==
   |   Indexed name (idx = 4)
   |     :authority
8b |     Literal value (len = 15)
   |     Huffman encoded:
db6d 883e 68d1 cb12 25ba 7f |     .m..h...%..
   |     Decoded:
   |     www.example.com
   | -> :authority: www.example\
   |     .com
```

Header Table (after decoding):

```
[ 1] (s = 57) :authority: www.example.com
[ 2] (s = 38) :path: /
[ 3] (s = 43) :scheme: http
[ 4] (s = 42) :method: GET
    Table size: 180
```

Decoded header set:

:method: GET


```
:scheme: http
:path: /
:authority: www.example.com
```

[E.3.2.](#) Second request

This request takes advantage of the differential encoding of header sets.

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
cache-control: no-cache
```

Reference set:

```
[ 1] :authority: www.example.com
[ 2] :path: /
[ 3] :scheme: http
[ 4] :method: GET
```

Hex dump of encoded data:

```
1b86 6365 4a13 98ff          | ..ceJ...
```

Decoding process:

```
1b          | == Literal indexed ==
            |   Indexed name (idx = 27)
```

```

86                                     |   cache-control
                                     |   Literal value (len = 8)
6365 4a13 98ff                       |   Huffman encoded:
                                     |   ceJ...
                                     |   Decoded:
                                     |   no-cache
                                     | -> cache-control: no-cache

```

Header Table (after decoding):

```

[ 1] (s = 53) cache-control: no-cache
[ 2] (s = 57) :authority: www.example.com
[ 3] (s = 38) :path: /
[ 4] (s = 43) :scheme: http
[ 5] (s = 42) :method: GET
      Table size: 233

```

Decoded header set:

```

cache-control: no-cache
:authority: www.example.com
:path: /
:scheme: http
:method: GET

```

[E.3.3.](#) Third request

This request has not enough headers in common with the previous request to take advantage of the differential encoding. Therefore, the reference set is emptied before encoding the header fields.

Header set to encode:

```

:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value

```

Reference set:

```
[ 1] cache-control: no-cache
[ 2] :authority: www.example.com
[ 3] :path: /
[ 4] :scheme: http
[ 5] :method: GET
```

Hex dump of encoded data:

```
8085 8c8b 8400 884e b08b 7497 90fa 7f89 | .....N..t.....
4eb0 8b74 979a 17a8 ff | N..t.....
```

Decoding process:

```
80 | == Empty reference set ==
   |     idx = 0
85 | == Indexed - Add ==
   |     idx = 5
   |     -> :method: GET
8c | == Indexed - Add ==
   |     idx = 12
   |     -> :scheme: https
8b | == Indexed - Add ==
   |     idx = 11
   |     -> :path: /index.html
84 | == Indexed - Add ==
   |     idx = 4
   |     -> :authority: www.example\
   |         .com
00 | == Literal indexed ==
88 |     Literal name (len = 10)
   |     Huffman encoded:
4eb0 8b74 9790 fa7f | N..t....
   |     Decoded:
   |     custom-key
89 |     Literal value (len = 12)
   |     Huffman encoded:
4eb0 8b74 979a 17a8 ff | N..t.....
   |     Decoded:
   |     custom-value
   |     -> custom-key: custom-valu\
   |         e
```

Internet-Draft

HPACK

December 2013

Header Table (after decoding):

```
[ 1] (s = 54) custom-key: custom-value
[ 2] (s = 48) :path: /index.html
[ 3] (s = 44) :scheme: https
[ 4] (s = 53) cache-control: no-cache
[ 5] (s = 57) :authority: www.example.com
[ 6] (s = 38) :path: /
[ 7] (s = 43) :scheme: http
[ 8] (s = 42) :method: GET
    Table size: 379
```

Decoded header set:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

[E.4.](#) Response Examples without Huffman

This section shows several consecutive header sets, corresponding to HTTP responses, on the same connection. `SETTINGS_HEADER_TABLE_SIZE` is set to the value of 256 octets, causing some evictions to occur.

[E.4.1.](#) First response

Header set to encode:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
0803 3330 3218 0770 7269 7661 7465 221d | ..302..private".
```

```

4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3120 474d 5430 1768 | 20:13:21 GMT0.h
7474 7073 3a2f 2f77 7777 2e65 7861 6d70 | ttps://www.examp
6c65 2e63 6f6d | le.com

```

Peon & Ruellan

Expires June 7, 2014

[Page 46]

Internet-Draft

HPACK

December 2013

Decoding process:

```

08 | == Literal indexed ==
   |   Indexed name (idx = 8)
   |     :status
03 |   Literal value (len = 3)
3330 32 | 302
   | -> :status: 302
18 | == Literal indexed ==
   |   Indexed name (idx = 24)
   |     cache-control
07 |   Literal value (len = 7)
7072 6976 6174 65 | private
   | -> cache-control: private
22 | == Literal indexed ==
   |   Indexed name (idx = 34)
   |     date
1d |   Literal value (len = 29)
4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3120 474d 54 | 20:13:21 GMT
   | -> date: Mon, 21 Oct 2013 \
   |   20:13:21 GMT
30 | == Literal indexed ==
   |   Indexed name (idx = 48)
   |     location
17 |   Literal value (len = 23)
6874 7470 733a 2f2f 7777 772e 6578 616d | https://www.exam
706c 652e 636f 6d | ple.com
   | -> location: https://www.e\
   |   xample.com

```

Header Table (after decoding):

```
[ 1] (s = 63) location: https://www.example.com
[ 2] (s = 65) date: Mon, 21 Oct 2013 20:13:21 GMT
[ 3] (s = 52) cache-control: private
[ 4] (s = 42) :status: 302
    Table size: 222
```

Decoded header set:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

[E.4.2.](#) Second response

The (":status", "302") header field is evicted from the header table to free space to allow adding the (":status", "200") header field to be copied from the static table into the header table.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set:

```
[ 1] location: https://www.example.com
[ 2] date: Mon, 21 Oct 2013 20:13:21 GMT
[ 3] cache-control: private
[ 4] :status: 302
```

Hex dump of encoded data:

```
848c                                     | ..
```

Decoding process:

```

84 | == Indexed - Remove ==
   |   idx = 4
   |   -> :status: 302
8c | == Indexed - Add ==
   |   idx = 12
   |   - evict: :status: 302
   |   -> :status: 200

```

Header Table (after decoding):

```

[ 1] (s = 42) :status: 200
[ 2] (s = 63) location: https://www.example.com
[ 3] (s = 65) date: Mon, 21 Oct 2013 20:13:21 GMT
[ 4] (s = 52) cache-control: private
    Table size: 222

```

Decoded header set:

```

:status: 200
location: https://www.example.com
date: Mon, 21 Oct 2013 20:13:21 GMT
cache-control: private

```

[E.4.3.](#) Third response

Several header fields are evicted from the header table during the processing of this header set. Before evicting a header belonging to the reference set, it is emitted, by coding it twice as an Indexed Representation. The first representation removes the header field from the reference set, the second one adds it again to the reference set, also emitting it.

Header set to encode:

```

:status: 200

```

```
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
location: https://www.example.com
content-encoding: gzip
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEIOU; max-age=3600; version=1
```

Reference set:

```
[ 1] :status: 200
[ 2] location: https://www.example.com
[ 3] date: Mon, 21 Oct 2013 20:13:21 GMT
[ 4] cache-control: private
```

Hex dump of encoded data:

```
8384 8403 1d4d 6f6e 2c20 3231 204f 6374 | .....Mon, 21 Oct
2032 3031 3320 3230 3a31 333a 3232 2047 | 2013 20:13:22 G
4d54 1d04 677a 6970 8484 8383 3a38 666f | MT..gzip.....:8fo
6f3d 4153 444a 4b48 514b 425a 584f 5157 | o=ASDJKHQKBZXOQW
454f 5049 5541 5851 5745 4f49 553b 206d | EOPIUAXQWEIOU; m
6178 2d61 6765 3d33 3630 303b 2076 6572 | ax-age=3600; ver
7369 6f6e 3d31 | sion=1
```

Decoding process:

```
83 | == Indexed - Remove ==
   | idx = 3
   | -> date: Mon, 21 Oct 2013 \
   | 20:13:21 GMT
```

```
84 | == Indexed - Remove ==
   | idx = 4
   | -> cache-control: private
84 | == Indexed - Add ==
   | idx = 4
   | -> cache-control: private
03 | == Literal indexed ==
   | Indexed name (idx = 3)
   | date
1d | Literal value (len = 29)
4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3220 474d 54 | 20:13:22 GMT
```


1d 04 677a 6970 84 84 83 83 3a 38 666f 6f3d 4153 444a 4b48 514b 425a 584f 5157 454f 5049 5541 5851 5745 4f49 553b 206d 6178 2d61 6765 3d33 3630 303b 2076 6572 7369 6f6e 3d31	- evict: cache-control: private -> date: Mon, 21 Oct 2013 20:13:22 GMT == Literal indexed == Indexed name (idx = 29) content-encoding Literal value (len = 4) gzip - evict: date: Mon, 21 Oct 2013 20:13:21 GMT -> content-encoding: gzip == Indexed - Remove == idx = 4 -> location: https://www.example.com == Indexed - Add == idx = 4 -> location: https://www.example.com == Indexed - Remove == idx = 3 -> :status: 200 == Indexed - Add == idx = 3 -> :status: 200 == Literal indexed == Indexed name (idx = 58) set-cookie Literal value (len = 56) foo=ASDJKHQKBZXO QWEOPIUAXQWEOIU; max-age=3600; version=1 - evict: location: https://www.example.com
---	--

	- evict: :status: 200 -> set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
--	--

Header Table (after decoding):

```
[ 1] (s = 98) set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWE0IU; max-age\  
    =3600; version=1  
[ 2] (s = 52) content-encoding: gzip  
[ 3] (s = 65) date: Mon, 21 Oct 2013 20:13:22 GMT  
    Table size: 215
```

Decoded header set:

```
cache-control: private  
date: Mon, 21 Oct 2013 20:13:22 GMT  
content-encoding: gzip  
location: https://www.example.com  
:status: 200  
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWE0IU; max-age=3600; version=1
```

[E.5.](#) Response Examples with Huffman

This section shows the same examples as the previous section, but using Huffman encoding for the literal values. The eviction mechanism uses the length of the decoded literal values, so the same evictions occurs as in the previous section.

[E.5.1.](#) First response

Header set to encode:

```
:status: 302  
cache-control: private  
date: Mon, 21 Oct 2013 20:13:21 GMT  
location: https://www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
0882 409f 1886 c31b 39bf 387f 2292 a2fb | ..@.....9.8."...  
a203 20f2 ab30 3124 018b 490d 3209 e877 | .. ..01$.I.2..w  
3093 e39e 7864 dd7a fd3d 3d24 8747 db87 | 0...xd.z.==$.G..  
2849 55f6 ff | (IU..
```

Decoding process:

```
08 | == Literal indexed ==
    | Indexed name (idx = 8)
    | :status
82 | Literal value (len = 3)
    | Huffman encoded:
409f | @.
    | Decoded:
    | 302
    | -> :status: 302
18 | == Literal indexed ==
    | Indexed name (idx = 24)
    | cache-control
86 | Literal value (len = 7)
    | Huffman encoded:
c31b 39bf 387f | ..9.8.
    | Decoded:
    | private
    | -> cache-control: private
22 | == Literal indexed ==
    | Indexed name (idx = 34)
    | date
92 | Literal value (len = 29)
    | Huffman encoded:
a2fb a203 20f2 ab30 3124 018b 490d 3209 | .... ..01$.I.2.
e877 | .w
    | Decoded:
    | Mon, 21 Oct 2013 20:13:21 \
    | GMT
    | -> date: Mon, 21 Oct 2013 \
    | 20:13:21 GMT
30 | == Literal indexed ==
    | Indexed name (idx = 48)
    | location
93 | Literal value (len = 23)
    | Huffman encoded:
e39e 7864 dd7a fd3d 3d24 8747 db87 2849 | ..xd.z.==$..(I
55f6 ff | U..
    | Decoded:
    | https://www.example.com
    | -> location: https://www.e\
    | xample.com
```

Internet-Draft

HPACK

December 2013

Header Table (after decoding):

```
[ 1] (s = 63) location: https://www.example.com
[ 2] (s = 65) date: Mon, 21 Oct 2013 20:13:21 GMT
[ 3] (s = 52) cache-control: private
[ 4] (s = 42) :status: 302
    Table size: 222
```

Decoded header set:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

[E.5.2.](#) Second response

The (":status", "302") header field is evicted from the header table to free space to allow adding the (":status", "200") header field to be copied from the static table into the header table.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set:

```
[ 1] location: https://www.example.com
[ 2] date: Mon, 21 Oct 2013 20:13:21 GMT
[ 3] cache-control: private
[ 4] :status: 302
```

Hex dump of encoded data:

848c

| ..

Internet-Draft

HPACK

December 2013

Decoding process:

```
84 | == Indexed - Remove ==
   |   idx = 4
   |   -> :status: 302
8c | == Indexed - Add ==
   |   idx = 12
   |   - evict: :status: 302
   |   -> :status: 200
```

Header Table (after decoding):

```
[ 1] (s = 42) :status: 200
[ 2] (s = 63) location: https://www.example.com
[ 3] (s = 65) date: Mon, 21 Oct 2013 20:13:21 GMT
[ 4] (s = 52) cache-control: private
    Table size: 222
```

Decoded header set:

```
:status: 200
location: https://www.example.com
date: Mon, 21 Oct 2013 20:13:21 GMT
cache-control: private
```

[E.5.3.](#) Third response

Several header fields are evicted from the header table during the processing of this header set. Before evicting a header belonging to the reference set, it is emitted, by coding it twice as an Indexed Representation. The first representation removes the header field from the reference set, the second one adds it again to the reference set, also emitting it.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
location: https://www.example.com
content-encoding: gzip
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWE0IU; max-age=3600; version=1
```

Reference set:

- [1] :status: 200
- [2] location: https://www.example.com
- [3] date: Mon, 21 Oct 2013 20:13:21 GMT
- [4] cache-control: private

Hex dump of encoded data:

```
8384 8403 92a2 fba2 0320 f2ab 3031 2401 | ..... ..01$.
8b49 0d33 09e8 771d 84e1 fbb3 0f84 8483 | .I.3..w.....
833a b3df 7dfb 36d3 d9e1 fcfc 3faf e7ab | ...}.6.....?...
fcfe fcbf af3e df2f 977f d36f f7fd 79f6 | ...../...o..y.
f977 fd3d e16b fa46 fe10 d889 447d e1ce | .w.=.k.F....D}..
18e5 65f7 6c2f | ..e.l/
```

Decoding process:

```
83 | == Indexed - Remove ==
   |   idx = 3
   |   -> date: Mon, 21 Oct 2013 \
   |       20:13:21 GMT
84 | == Indexed - Remove ==
   |   idx = 4
   |   -> cache-control: private
84 | == Indexed - Add ==
   |   idx = 4
   |   -> cache-control: private
```

```

03 | == Literal indexed ==
    | Indexed name (idx = 3)
    |   date
92 | Literal value (len = 29)
    | Huffman encoded:
a2fb a203 20f2 ab30 3124 018b 490d 3309 | .... ..01$.I.3.
e877 | .w
    | Decoded:
    | Mon, 21 Oct 2013 20:13:22 \
    | GMT
    | - evict: cache-control: pr\
    |   ivate
    | -> date: Mon, 21 Oct 2013 \
    |   20:13:22 GMT
1d | == Literal indexed ==
    | Indexed name (idx = 29)
    |   content-encoding
84 | Literal value (len = 4)
    | Huffman encoded:
e1fb b30f | ....

```

```

    | Decoded:
    | gzip
    | - evict: date: Mon, 21 Oct\
    |   2013 20:13:21 GMT
84 | -> content-encoding: gzip
    | == Indexed - Remove ==
    |   idx = 4
    | -> location: https://www.e\
    |   xample.com
84 | == Indexed - Add ==
    |   idx = 4
    | -> location: https://www.e\
    |   xample.com
83 | == Indexed - Remove ==
    |   idx = 3
    | -> :status: 200
83 | == Indexed - Add ==
    |   idx = 3
    | -> :status: 200
3a | == Literal indexed ==
    | Indexed name (idx = 58)

```

<pre> b3 df7d fb36 d3d9 e1fc fc3f afe7 abfc fefc bfaf 3edf 2f97 7fd3 6ff7 fd79 f6f9 77fd 3de1 6bfa 46fe 10d8 8944 7de1 ce18 e565 f76c 2f </pre>	<pre> set-cookie Literal value (len = 56) Huffman encoded: .}.6.....?..... .../...o..y..w. =.k.F....D}.....e .l/ Decoded: foo=ASDJKHQKBZXOQWEOPUIUAXQ\ WEOIU; max-age=3600; versi\ on=1 - evict: location: https://\ /www.example.com - evict: :status: 200 -> set-cookie: foo=ASDJKHQ\ KBZXOQWEOPUIUAXQWEIOIU; ma\ x-age=3600; version=1 </pre>
---	--

Header Table (after decoding):

```

[ 1] (s = 98) set-cookie: foo=ASDJKHQKBZXOQWEOPUIUAXQWEIOIU; max-age\
=3600; version=1
[ 2] (s = 52) content-encoding: gzip
[ 3] (s = 65) date: Mon, 21 Oct 2013 20:13:22 GMT
Table size: 215

```

Decoded header set:

```

cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
content-encoding: gzip
location: https://www.example.com
:status: 200
set-cookie: foo=ASDJKHQKBZXOQWEOPUIUAXQWEIOIU; max-age=3600; version=1

```

Authors' Addresses

Roberto Peon
Google, Inc

E-Mail: fenix@google.com

Herve Ruellan
Canon CRF

E-Mail: herve.ruellan@crf.canon.fr