

Individual submission  
Internet-Draft  
Intended status: Informational  
Expires: May 1, 2020

W. Handte  
Facebook, Inc.  
October 29, 2019

Security Considerations Regarding Compression Dictionaries  
draft-handte-httpbis-dict-sec-00

Abstract

Dictionary-based compression enables better performance, but brings state into the process of compression, with all the complexities that follow. This document explores the security implications of this technique in the context of internet protocols and enumerates known risks and mitigations.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 1, 2020.

Copyright Notice

Copyright (c) 2019 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 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 . . . . .	3
2. Basis . . . . .	3
2.1. Compression Environments . . . . .	3
2.2. Security Properties . . . . .	4
2.3. Threat Model . . . . .	4
2.4. Existing Attacks . . . . .	4
3. Dictionaries . . . . .	5
3.1. Dictionary Compression . . . . .	5
3.2. Dictionary Contents . . . . .	5
3.2.1. Unstructured Dictionaries . . . . .	5
3.2.2. Structured Dictionaries . . . . .	6
3.3. Using Dictionaries . . . . .	6
3.3.1. Generating Dictionaries . . . . .	6
3.3.2. Identifying Dictionaries . . . . .	7
3.3.3. Distributing Dictionaries . . . . .	9
3.3.4. Selecting Dictionaries . . . . .	9
3.3.5. Using Dictionaries . . . . .	10
3.3.6. Deleting Dictionaries . . . . .	10
4. Risks . . . . .	11
4.1. Revealing Message Content . . . . .	11
4.1.1. By Observing Which Dictionary is Used . . . . .	11
4.1.2. By Observing Message Size . . . . .	12
4.1.3. By Observing Timing . . . . .	13
4.2. Revealing Dictionary Content . . . . .	14
4.2.1. By Observing Message Size . . . . .	14
4.2.2. In Compression . . . . .	14
4.2.3. In Decompression . . . . .	14
4.3. Manipulating Message Content . . . . .	15
4.3.1. By Manipulating Message Content . . . . .	16
4.3.2. By Manipulating Dictionary Content . . . . .	16
4.3.3. By Manipulating Dictionary Identifiers . . . . .	17
4.4. Obfuscating Message Content . . . . .	17
4.4.1. From Intermediaries . . . . .	17
4.4.2. Multiple Representations . . . . .	18
4.5. Tracking Users . . . . .	18
4.5.1. Through Dictionary Negotiation . . . . .	18
4.5.2. Through Dictionary Retrieval . . . . .	19
4.6. Denial of Service . . . . .	19
4.7. Resource Exhaustion . . . . .	19
4.7.1. Resources . . . . .	19
4.7.2. Targets . . . . .	22
4.8. Generating Dictionaries . . . . .	24
4.8.1. Handling Samples . . . . .	24
4.8.2. Tagging Mitigations . . . . .	24
4.8.3. Probabilistic Mitigations . . . . .	25
4.9. Complexity . . . . .	25

5. Conclusions . . . . .	25
6. IANA Considerations . . . . .	26
7. Security Considerations . . . . .	26
8. References . . . . .	26
8.1. Normative References . . . . .	26
8.2. Other Examples of Dictionary-Like Compression . . . . .	26
8.3. Informative References . . . . .	27
Appendix A. Acknowledgements . . . . .	30
Author's Address . . . . .	30

## 1. Introduction

General-purpose data compression algorithms are designed to achieve good performance on many different kinds of data. However, that general-purpose nature makes them, to a certain extent, jacks of all trades and masters of none: a compressor that has been tuned for a specific use case can always perform better than a generic equivalent.

In response, a number of modern compression algorithms (including DEFLATE [DEFLATE], Brotli [BROTLI], and Zstandard [ZSTD]) have developed a generic capability to specialize themselves. In addition to the actual message to be processed, these compressors allow users to provide additional context information, which the compressor and decompressor can use to tailor their internal states to that particular use case. To the extent that this auxiliary data matches the nature of the message being compressed, the compressor can use it to produce a smaller compressed representation of the message. This auxiliary data can include various things, but it has come to be known as a "dictionary."

As dictionary-based compression has been adopted, it has been found that its use can present security challenges. This document is a collection of those challenges. As future use cases for dictionaries are contemplated, this document can be used as a checklist to ensure that the protocols, their specifications, and their implementations have been appropriately evaluated against these concerns.

## 2. Basis

### 2.1. Compression Environments

The security of any use of compression depends greatly on the environment in which it is deployed, and the threats it is subjected to. This document analyzes dictionary-based compression as it might be used by a generic internet protocol, in which:

- o Agents exchange messages over possibly-trusted, possibly-authenticated, possibly-encrypted channels, which are vulnerable to some combination of traffic analysis, eavesdropping, and manipulation.
- o Agents exchange messages with parties they may not trust.
- o Agents may take protocol actions (generating, sending, receiving, and interpreting messages) in response to triggers other than user action. Some examples include:
  - \* Replying automatically to received messages.
  - \* Relaying or forwarding received messages to other agents (e.g., an SMTP relay).
  - \* Exchanging messages at the behest of trusted or untrusted code (e.g., trusted: a website codebase generating responses to HTTP requests, untrusted: a website's JavaScript code running in a browser).

This document aims to enumerate all security risks raised when using dictionary-based compression in this baseline environment. In addition to attempting an exhaustive list of possible security risks, this document will identify desirable properties of the protocol stack and environment in which the compression is used and other methods with which individual concerns can be obviated or mitigated.

## 2.2. Security Properties

[TODO]

## 2.3. Threat Model

[TODO]

## 2.4. Existing Attacks

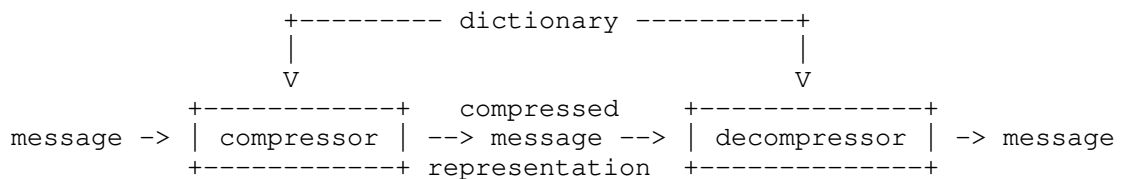
This document excludes from its analysis security risks that are already present without the use of dictionary compression.

In particular, compression as it broadly used today--without dictionaries--is known to introduce vulnerabilities. The most well known series of these attacks ([CRIME] et al.) recovers message content of inaccessible or encrypted traffic by observing message sizes while manipulating other parts of the message or traffic stream.

### 3. Dictionaries

#### 3.1. Dictionary Compression

Classically, compression algorithms operate as stateless, pure functions. In that mode, their output depends solely on the input message and the algorithm's implementation details. Dictionaries break that paradigm, introducing an additional input to the compression and decompression operations. Compressors may then leverage the contents of that additional input--the dictionary--to produce more compact representations of their inputs.



In introducing this other element, the interpretation of the compressed message becomes dependent on the content of the dictionary, and therefore same dictionary that was used to compress a message must be presented at decompression time. In this way, dictionaries are in effect an out-of-band communication or pre-shared key between the compressor and decompressor.

#### 3.2. Dictionary Contents

In principle, the contents of a dictionary are solely the concern of the compressor and decompressor, and implementations should be free to treat them as opaque blobs. However, when analyzing their security characteristics, it's useful to understand the data that is actually present in a dictionaries.

Dictionaries take two broad forms.

##### 3.2.1. Unstructured Dictionaries

Some compressors (e.g., DEFLATE [DEFLATE] and Zstandard [ZSTD]) accept arbitrary, unstructured bytestreams as dictionaries. In these cases, the dictionary is used purely as a buffer in which LZ77-style content matches can be found [LZ77]. That is, when the dictionary contains some sequence of bytes that is also present in the message, the compressor can choose to represent those bytes by referencing them in the dictionary, rather than by representing them literally.

### 3.2.2. Structured Dictionaries

Some compressors (e.g., Brotli [BROTLI] and Zstandard [ZSTD]) accept dictionaries that conform to a specific and defined format. In these cases, the dictionary data can consist of multiple components, each of which is used in different ways.

**metadata:** The dictionary may contain metadata that identifies the dictionary. For example, Zstandard dictionaries include a 32-bit integer ID field.

**statistics:** The dictionary may contain frequency distributions of various kinds of symbols, which the compressor can use to more efficiently encode the corresponding streams instead of using a default frequency distribution.

**initial values:** For example, Zstandard allows the dictionary to initialize certain parts of the compressor's internal state (in particular, the initial values of Repeated\_Offset1, Repeated\_Offset2, and Repeated\_Offset3) [ZSTD].

**instructions:** The dictionary may describe preprocessing or transformation steps to be taken on the input. [TODO: expand]

**corpus content:**

**untokenized:** For LZ77-style compressors [LZ77], the structured dictionaries may still contain unstructured content for the compressor to make matches against.

**tokenized:** Alternatively, for LZ78-style compressors [LZ78], the match content is tokenized (i.e., it consists of a collection of independent strings, serialized in some form).

### 3.3. Using Dictionaries

In order to use compression dictionaries in a system, it is not only the internals and integration points of the compressor and decompressor whose behavior must change. Dictionaries make compression stateful, and applications that use dictionaries must therefore participate in the whole lifecycle of state management.

#### 3.3.1. Generating Dictionaries

As noted in Section 3.2.1, some compression algorithms can accept arbitrary, unstructured inputs as dictionaries. These unstructured dictionaries do not require an explicit generation step; users can simply repurpose existing messages as dictionaries. This potentially

avoids the need to perform additional coordination and communication to distribute purpose-built dictionaries. See for example the Compression Dictionaries for HTTP/2 proposal [I-D.vkrasnov-h2-compression-dictionaries].

Alternatively, the dictionary may be a separate object, purpose-built for the task. Generating such a dictionary may be desirable for a number of reasons, including:

- o Building a dictionary is necessary to produce the structure in a structured dictionary.
- o Trained dictionaries generally perform better than using raw content. The training process selects the parts of the sample corpus that are useful for compression and discard the parts that are not, producing a more compact and more effective dictionary.
- o The training process is an opportunity to sanitize the content that ends up being used as a dictionary, potentially enhancing security and privacy (see Section 4.8).

In general, an algorithm is run over a corpus of sample messages (such as the COVER algorithm [COVER] in Zstandard), which selects commonly occurring substrings and bundles them together.

Any structured metadata (e.g., symbol distribution statistics) can then be calculated. For example, Zstandard then compresses some of the sample messages it was given with the dictionary and aggregates the statistics resulting from those compressions and writes them into the dictionary's header.

### 3.3.2. Identifying Dictionaries

If freedom exists in a system as to which dictionary is to be used for a given message, there must be some way to distinguish which dictionary to use, so that decompressors can use the same one. In practice, this means associating each dictionary with an identifier.

Popular methods to do this include:

Identity ID: The "identifier" for the dictionary is the dictionary itself. This is not really very popular, since information theory strongly suggests that a compressed message without a dictionary will always be smaller than a message compressed with a dictionary plus the dictionary.

Arbitrary IDs: The scheme associates an arbitrary identifier (e.g., a number or string) with this dictionary. This can have the

advantage of being the most compact, but has the disadvantage that it neither describes the content of the dictionary nor where to get it.

**Content-Derived IDs:** Identifiers that are deterministically derived from the content they identify (such as hashes), when designed well, have the benefit that they can validate the associated dictionary without requiring trusting the dictionary source. (Though they are of course vulnerable to collision attacks.) They have the disadvantage that they do not describe where to source the dictionary. In order to be secure, they may also have to be relatively verbose.

**Location-Based IDs:** Identifiers of this form (notably, URLs) do not identify the content directly, but rather describe where to get it. They are suitable insofar as that source can be trusted to reliably serve the same content to different participants.

#### 3.3.2.1. Existing Systems

Existing compression schemes have selected the following identification systems:

**DEFLATE:** DEFLATE writes an Adler32 checksum of the dictionary into its compressed message header and checks it at decompression-time.

**Brotli:** Brotli always implicitly uses a single static dictionary. As such, no identifier is needed or provided [BROTLI].

**Shared Brotli:** Shared Brotli uses either a 256-bit Highwayhash digest of the dictionary or a direct pointer to the dictionary when it is included in the same compressed stream [I-D.vandevenne-shared-brotli-format].

**Zstandard:** Zstandard uses 32-bit integers to identify dictionaries [ZSTD].

**SDCH:** SDCH uses a URL to describe how to fetch a dictionary and then a hash (a 96-bit prefix of the SHA-256 digest of the dictionary) in negotiations [I-D.lee-sdch-spec].

**CDH2:** Compression Dictionaries for HTTP/2 uses an 8-bit integer [I-D.vkrasnov-h2-compression-dictionaries].



### 3.3.3. Distributing Dictionaries

Dictionaries must themselves be made accessible to participants. There are several possible approaches to doing this:

**static:** The protocol defines the set of dictionaries. Protocol implementations can statically include or independently generate these dictionaries. No further distribution mechanism is required.

**local:** When dictionaries are not specified by the protocol, but are derived locally or provided by the user, no dictionary distribution mechanism is required, although a negotiation mechanism might be.

**centralized:** The set of dictionaries in use by the system changes over time, coordinated by and available from a central authority.

**distributed:** The set of dictionaries in use by the system changes over time. Some or all participants can generate and publish dictionaries.

### 3.3.4. Selecting Dictionaries

Related to the above, because the same dictionary must be used to compress and decompress a particular message, it is necessary for the compressor and decompressor to come to an understanding as to which dictionary they will use for a given message, presumably based on selecting which dictionary of those available to both the sender and receiver is most suitable. This selection process can take multiple forms:

**implicit:** In situations where the compressor or protocol specifies a single dictionary that is always used (e.g., Brotli [BROTLI]), no particular selection process is required. Use of the compression scheme at all (which may or may not itself be negotiated) is sufficient to identify the dictionary to use.

**unilateral:** When the set of dictionaries available to the decompressing agent is known to the compressing agent, the compressing agent may unilaterally select a dictionary to use, and include an identification of that dictionary in either the compressed message itself (e.g., Zstandard's Dictionary\_ID field in the frame header) or in protocol metadata (e.g., an HTTP response header). This mechanism can be applied in simple situations, such as when the set of dictionaries used by the protocol is fixed and guaranteed to be immediately available to all participants (such as by being included in the

implementation's installation). It can also be applied to a more loose definition of availability, if the decompressing agent is known to be capable of retrieving the dictionary based on the provided identifier, even if it doesn't have the dictionary at present.

**bilateral:** When the set of dictionaries available to each party is not known to the other, additional messages may be required in order for the compressing agent to select a dictionary available to both both parties. In particular, while other negotiation patterns only require a flow of information from the compressor to the decompressor, which matches the flow of the compressed message itself, this mechanism requires communication in both directions.

### 3.3.5. Using Dictionaries

Having selected and retrieved a dictionary, it remains to actually present the dictionary to the compressor or decompressor and perform the compression operation.

Dictionaries, whether structured or not, are flat byte streams. In order to be used (especially in compression), most implementations require that a preparation step be performed on the content of the dictionary, populating the compressor's internal datastructures.

This materialization of the dictionary can sometimes be performed transparently as part of the compression or decompression operation. Alternatively, some compressors allow this materialization step to be performed separately / explicitly. When this capability is used, the work of processing the serialized dictionary into the compressor's internal datastructures only needs to be performed once, even when this materialized dictionary object is used for many compressions or decompressions. This can lead to significant efficiencies.

### 3.3.6. Deleting Dictionaries

Dictionary compression inherently entangles the lifetimes of different pieces of data. When a dictionary is generated, it collects and incorporates information about the data it was trained on (whether that be diffuse statistical information, small common substrings or tokens, or significant contiguous excerpts of the training data). When that dictionary is used to compress a set of messages, it must be retained by the system for as long as the system desires to be able to decompress any of those messages. The lifetime of information derived from individual messages is thus tied to the lifetime of many messages, or even the whole system. This introduces complexities for systems that wish to minimize or bound the lifetime of individual pieces of data.

## 4. Risks

These subsections each describe a class of security issues that have been raised concerning dictionary-based compression and the surrounding protocol mechanisms. Where possible and known, mitigations are described.

### 4.1. Revealing Message Content

This section discusses attacks that use dictionary compression to recover content in the message.

#### 4.1.1. By Observing Which Dictionary is Used

Because dictionaries' effectiveness improves the more that they target a specific type of data, a protocol may want to use multiple dictionaries, each targeting a subclass of the system's traffic. Alternatively, a participant may always avoid using a dictionary in certain scenarios, such as when reporting an error. When this is the case, the use of a particular dictionary or not for a message implies that the message belongs to the corresponding subclass of traffic.

The metadata identifying which dictionary was used to compress a message should therefore be protected to the same extent that the message content is protected. (Similarly, the choice of dictionary and any data exchanged in that selection process may reveal other information about the sender and receiver, independent of the content of the specific message being handled, which is discussed in Section 4.5.1.)

This information may itself be inferred from other signals, and therefore serve as a stepping stone connecting those signals to conclusions about message content.

**Message Size:** Observations of message sizes, especially headers or connection negotiations (also discussed in Section 4.1.2), can indicate whether a dictionary was used, or even perhaps which dictionary was used.

**Timing:** Compression with and without a dictionary may take observably different amounts of time. This is also discussed in Section 4.1.3.

**Dictionary Retrieval:** When dictionaries are retrieved dynamically, another vector for learning this information is simply observing whether a message triggers a fetch for a dictionary, and if so, which dictionary. (This is also discussed in Section 4.5.2.) Protocols should consider decoupling retrieving dictionaries

(especially when doing so is easily observable) from using them. For example, SDCH advertises and retrieves dictionaries independently of using them [I-D.lee-sdch-spec].

#### 4.1.2. By Observing Message Size

By manipulating a portion of the message and observing the overall size of the compressed message, the attacker can recover information about the portions of the message not under its control [BREACH] [CRIME] [HEIST]. Given that dictionary-based compression is an extension of dictionary-less compression, it is certainly also vulnerable to this attack.

In particular, the dictionary itself can be used in this sort of attack, to the extent that its contents are attacker-controlled. Note that the ability to control which dictionary is used may indirectly give an attacker the effective ability to modify the contents of the dictionary.

Protocol designers should therefore prevent parties that will not have access to the message content from being able to influence the dictionary used to compress the message.

In settings where the dictionary that is used is derived from previous traffic, especially if previous traffic is directly used as a dictionary, the problem of ensuring that private data and attacker-controlled data grows in complexity. In such a scheme, the attacker may also be able to exercise more control over the content of the dictionary if they can influence the order in which messages are exchanged. Protocols of this sort may wish to place strong controls on the kinds of messages that can be included in the dictionary. See for example [I-D.vkrasnov-h2-compression-dictionaries].

The remaining question is whether the dictionary constitutes a third class of data (fixed, known data), with distinct security properties. That is, even if the dictionary is neither under attacker control nor does it contain private information, can its use still reveal information about the contents of the message under compression.

##### 4.1.2.1. Mitigating with Padding

One possible mitigation of the compressed message size oracle is to add padding to messages, either at the compression level or at the transport layer (e.g., [I-D.pironti-tls-length-hiding]). Even simple padding schemes can significantly inflate the cost of mounting such an attack, if not mitigate it completely.

#### 4.1.2.2. Mitigating by Separating Content

Another possible strategy to mitigate this attack is to avoid letting attacker-controlled data be matched against private data. This can be accomplished by avoiding compressing one or the other, or by compressing them independently of each other. See, e.g., [CLOUDFLARE-NO-COMPRESS].

#### 4.1.2.3. Mitigating by Avoiding Repeated Compressions

A crucial feature of these attacks is that they require the message under attack to be re-compressed many times (proportional to the amount of information being extracted). The attack can therefore be mitigated either by limiting the number of times the same message can be compressed (rate-limiting), or by making sure that it is not the same message that is compressed every time.

That is to say, these attacks are most effective when the attacker-controlled data is the only thing that is changing between compressions. Changing or randomizing content (ideally, including the secrets in question) in the message on each compression can make it much harder to extract information.

#### 4.1.3. By Observing Timing

Timing is another classic side-channel through which information can leak. An attacker could potentially observe the time taken during compression or decompression, and draw conclusions about the contents of a message. As discussed in Section 4.7.1.3.1, it is possible that a dictionary could affect the efficiency of compression and decompression.

In addition, timing can act as a vector for extracting information from another side-channel. As described in the HEIST attack [HEIST], compression ratio information can be leaked by counting round-trip latencies.

Alternatively, while compression and decompression are usually relatively fast and fairly content-insensitive operations, retrieving and initializing a dictionary might be a high-latency operation, and therefore may be identifiable by observing timing. Timing is therefore another potential avenue to observe which dictionary is used, which may in turn reveal information about the message being processed (Section 4.1.1).

## 4.2. Revealing Dictionary Content

This section investigates the ability to leverage dictionary-based compression to reveal data other than the message content being compressed (i.e., revealing content used as the dictionary). Note that this is only of interest when there are secrets in the dictionary, which violates the common model that is mostly analyzed in this document, in which the dictionary is assumed to be a shared, public resource.

In systems with multiple privacy domains, the ability to nominate arbitrary resources in that system as dictionaries poses a risk.

Protocol designers and implementors should ensure that compressing and decompressing agents cannot use as dictionaries resources from privacy domains that either agent does not have access to.

A corollary is that a transport system that mixes resources from multiple privacy domains into the same compression context through dictionary-based compression should not reveal the compressed representation of messages (or information derived from the compressed representation, such as its size) to other components of the system that are only trusted in a particular privacy domain.

### 4.2.1. By Observing Message Size

Analogously to Section 4.1.2, an attacker can exploit knowledge about the contents of a message and its compressed size to draw conclusions about the contents of the dictionary.

### 4.2.2. In Compression

If an attacker can inspect the compressed representation of a message, they may be able to draw conclusions about the contents of the dictionary that was used to compress it. This is especially the case if the attacker knows the original message that was compressed (i.e., a known-plaintext attack) or if the attacker can supply the message to be compressed (i.e., a chosen-plaintext attack), and is helped if the attacker can cause the message to be compressed multiple times while varying some aspect of the compression.

### 4.2.3. In Decompression

In compression schemes that support the use of dictionaries, and especially unstructured dictionaries, it is possible to craft compressed messages independent of a dictionary in such a way that, when decompressed with a provided dictionary, the decompressed message that is produced will reveal information about the contents

of the dictionary that was not known by the compressor (possibly trivially, by directly reproducing some or all of the dictionary's contents).

Consider a protocol that allows a compressing agent to freely identify any other resource in the system as the dictionary for a message. The compressing agent could select as a dictionary some resource to which the decompressing agent has access, but to which it does not. Without access to that resource, it could nonetheless generate a compressed message the effect of which would be to reproduce that resource in part or in its entirety. This message, decompressed by the target, would cause a resource in the compressing agent's trust domain to appear to have the contents of a resource it does not itself have access to.

This could cause the decompressing agent to take some action that the compressing agent would not otherwise have had the authority to initiate. Alternatively, with some additional mechanism, the compressing agent could then cause the decompressing agent to reveal the uncompressed message (i.e., the selected third-party resource) back to the compressing agent.

#### 4.3. Manipulating Message Content

When the decompressing agent uses a different dictionary to decompress a message than was used to compress the message (which is possible due to confusion on the part of either the compressing or decompressing agent), the reconstituted message produced by decompression may differ from the original message the compressing agent intended.

An attacker that can induce this situation can therefore use dictionary compression to manipulate the perceived content of messages, even when they cannot directly manipulate the contents of the messages themselves.

A particular implication of this is that a compressed message may have multiple interpretations. In one context (with one dictionary), the message can be constructed so as to appear benign or to pass a validation or authentication step when decompressed. Later, if a different component or agent can be induced to decompress the same message with a different dictionary, the reconstructed message may be completely different.

A general mitigation against this attack is to specify mechanisms to validate the integrity of the message. In particular, it may be desirable to validate the ultimate, uncompressed message, rather than validating the various components that the decompressing agent relies

on to reconstitute the uncompressed message--the compressed message, the metadata identifying the dictionary, the associated dictionary contents, etc. (However, this has its own problems [ENCRYPT-THEN-AUTHENTICATE].)

#### 4.3.1. By Manipulating Message Content

The degenerate version of this attack is to manipulate the uncompressed message by directly manipulating the compressed representation of the message. In such a scenario, the presence or absence of a dictionary is irrelevant. In most cases, this attack is defended against by some scheme that protects the integrity of the compressed message.

However, it is useful to point this attack out, as the other attacks in this space aim to achieve the same result indirectly, and may do so by exploiting protocols which protect the integrity of the compressed message, but perhaps not its metadata describing which dictionary to use nor the contents of that dictionary, such as might arise particularly if dictionary-based compression is an extension to an existing protocol.

#### 4.3.2. By Manipulating Dictionary Content

One possible avenue for this kind of attack is to cause the compressing agent and decompressing agent to have differing views of the same dictionary (whether by manipulating a participant's local copy or by causing a fetch to return different results for different users or otherwise).

Protocol designers should therefore take care to protect the integrity of dictionaries. Two broad strategies exist to do so.

##### 4.3.2.1. Mitigating by Validating Dictionary Contents

In the first, the identifier for the dictionary may itself be used to validate the contents that are retrieved, if the identifier scheme includes a cryptographically secure digest of the identified dictionary's contents (see Section 3.3.2). Alternatively, even if the identifier itself does not provide for , designers should specify other mechanisms to ensure the integrity and correctness of dictionaries (signatures, checksums, etc.). See for example schemes like Subresource Integrity [SRI].



#### 4.3.2.2. Mitigating by Validating Dictionary Sources

Alternatively, participants can rely on a secure chain of custody from a trusted source. ... [TODO]

In practice, it is probably advisable to implement both mitigations in some form.

#### 4.3.3. By Manipulating Dictionary Identifiers

Another similar attack is to cause the different agents to have differing views of which dictionary to use. That is, even if the integrities of compressed messages and dictionary contents are protected, if the association between one and the other can be manipulated, the same effect can be achieved.

#### 4.4. Obfuscating Message Content

This section discusses attacks that obfuscate a malicious response's content through the use of dictionary-based compression.

##### 4.4.1. From Intermediaries

Various internet protocols exchange messages through intermediaries which inspect or modify the traffic as it passes by (proxies, caches, firewalls, etc.), sometimes for reasons that include security. If the compressing and decompressing agents on a connection use a dictionary to compress the messages they exchange, and the intermediaries between them are not themselves capable of processing messages compressed this way, the intermediaries may be prevented from being able to inspect the traffic, which may harm their ability to detect and filter malicious traffic.

In practice, the relevance of this concern is questionable. Intermediaries of this form [PERVASIVE-MONITORING] can be more harmful than they are beneficial to the security of participants and their traffic. Many protocols are moving towards end-to-end encrypted models that preclude intermediaries from interacting with messages in this way.

Nonetheless, designers of protocols that involve intermediaries that might not support dictionary based compression should give those intermediaries the ability to downgrade the message exchange to not use dictionaries. Intermediaries which inspect messages in the course of their business should either implement the dictionary based compression scheme in question or downgrade the message exchange to avoid its use.

#### 4.4.2. Multiple Representations

Although the majority (if not the entirety) of compression schemes do not guarantee determinism in compression, many implementations are deterministic in practice (under fixed parameters). Experience has demonstrated that this state of affairs sometimes entices implementors into confusing equality-of-message comparison with equality-of-representation comparison. Representing the same message in a new way can therefore violate assumptions and potentially be used as a vector for exploitation. Dictionaries potentially contribute to this issue, by introducing a new vector for non-determinacy in the compressed representation of a message.

Users of compression should therefore avoid assumptions that a message will always be transformed into the same compressed representation.

#### 4.5. Tracking Users

This section discusses attacks that identify users through their negotiation and use of dictionaries.

Like any other protocol extension or option, the use or advertisement of dictionaries, may allow observers to distinguish participants that do and do not support the feature.

##### 4.5.1. Through Dictionary Negotiation

In systems which distribute dictionaries dynamically, a participant or observer may be able to learn about the past actions of other participants by observing the dictionaries they advertise or select.

For example, if a user exchanged messages with some site ([www.mybank.com](http://www.mybank.com)), and in doing so acquired dictionaries published by that operator, and then sometime later negotiated a connection with some other site ([www.curiousaboutyou.com](http://www.curiousaboutyou.com)), in which the user advertised the dictionaries in their possession, the second operator could reasonably conclude that the user had a bank account at MyBank.

Designers of protocols that use dynamically distributed and negotiated dictionaries should therefore take care that dictionaries distributed in one privacy domain are not advertised or used in others without reason.

#### 4.5.2. Through Dictionary Retrieval

The distributor of a dictionary may also be able track the propagation of traffic amongst participants as it receives requests for a particular dictionary, especially if it can collude with the party that generated that message to use a unique dictionary identifier.

Dictionaries that are dynamically fetched should therefore be fetched from the same privacy domain they are used in.

#### 4.6. Denial of Service

Because dictionary-based compression introduces additional dependencies to the processes of generating and interpreting messages, an attacker that cause those dependencies to be unavailable can potentially cause participants to fail to process messages.

Protocols that use dictionary-based compression, especially when the dictionaries are retrieved in ways that could fail, should be prepared to gracefully degrade when those fetches fail. Designers may consider whether messages should only be compressed with dictionaries known to already be in the possession of the recipients.

#### 4.7. Resource Exhaustion

This section discusses attacks that use dictionaries and dictionary-based compression to induce failures through the exhaustion of various resources.

Aside from more specific concerns and corresponding protections discussed in the following sections, implementors should take care to apply at least the same resource usage constraints to dictionaries that they do to the other traffic they handle. Stronger constraints may be warranted, in fact, since the goal of dictionaries is to lower total resource consumption.

##### 4.7.1. Resources

###### 4.7.1.1. Bandwidth

Attacks of this form cause the target to consume their network resources, resulting in expense and degradation of service.

#### 4.7.1.1.1. Messages

If dictionaries can be used to make the compressed representation of messages artificially large, it may be possible to cause normal traffic to consume disproportionately large bandwidth. With existing dictionary schemes, this is unlikely.

The reverse is also potentially dangerous, though. Systems that are accustomed to using dictionary-based compression (and whose resources are allocated according to the efficiencies achieved thereby) may be vulnerable to resource exhaustion when subjected to downgrade attacks. If an attacker can force the system to fall back to not using dictionaries, or to using bad dictionaries, or to not using compression at all, the system may exceed its allocated network resources.

#### 4.7.1.1.2. Dictionaries

In protocols in which dictionaries are distributed dynamically, it may be possible to cause a target to repeatedly attempt to fetch dictionaries, whether by causing dictionary fetches to fail, triggering retries, or by causing the target to use many new dictionaries that it must then load.

Since dictionaries can be quite large relative to the messages they are used to compress, this could potentially be an effective amplification attack.

#### 4.7.1.2. Storage

Attacks of this form target the storage resources of a participant (any of main memory, cache, disk space, etc.).

##### 4.7.1.2.1. Message Size

The same concerns apply here as in Section 4.7.1.1.1.

Additionally, if dictionaries can be used to make the compressed representation of a message extremely small relative to the its uncompressed size, they may play a role in enabling a "zip bomb" type attack, in which a specially crafted, small (and therefore cheap to send) message causes the recipient to consume a huge amount of storage space after decompression.

Implementors should therefore apply storage quotas to messages based on the size of the representation in which they will actually be stored. Implementors may also wish to consider rejecting messages

whose compressed representation is significantly larger than the message represented.

#### 4.7.1.2.2. Message Duplication

Obviously, flooding a target with messages is an easy way exhaust that participant's resources. Using a dictionary does not natively affect that brute force strategy. However, simple mitigations to this sort of attack sometimes leave chinks in systems' armor, which dictionaries might play a role in exploiting.

For example, if an attacker can cause a participant to receive and store a single logical message more than once, with different metadata (such as the dictionary used) or with a different compressed representation (as a result of using a different dictionary), the participant may not be able or willing to deduplicate the message. For example, an HTTP Cache may be forced to store the same resource multiple times, compressed with different dictionaries, if the choice of dictionary is part of the cache's secondary key [HTTP-CACHING].

#### 4.7.1.2.3. Dictionaries

Another possible avenue of attack would be to cause a participant to consume space by storing the dictionaries themselves. The effectiveness of attacks of this form are driven by the product of (1) the number of dictionaries stored, (2) their size, and (3) how long they are retained.

Dictionaries may themselves be fairly large. But one thing to note in particular is that, when in use, the space consumed by a dictionary may be significantly greater than its raw size. In order to be used in compression or decompression (but particularly in compression), the dictionary contents must be loaded into the compressor's internal datastructures. This can be done at compression-time, for every compression, using the datastructures already allocated for that compression.

Alternatively, some compression algorithms allow the user to do this preparation step separately, producing a materialized representation of the dictionary in memory that can be reused across a number of compression operations (e.g., a ZSTD\_CDict). While this avoids duplicated work (processing the dictionary for each compression), applications which cache these materialized dictionaries can accidentally consume a lot of memory. In addition to the factors mentioned above that control the total size of stored dictionaries, the expansion factor as those dictionaries are materialized is controlled by the compression settings (and potentially instructions in the dictionary).

Applications that allow other participants to influence the contents, number, size, retention period, or compression settings of dictionaries should take care to constrain the total at rest and in-memory footprints of those dictionaries.

#### 4.7.1.3. Computation

Attacks of this form target the computational resources (and by extension, the time and energy) of a participant in the protocol.

##### 4.7.1.3.1. Using a Dictionary

For existing compressors that support dictionaries, compression and decompression with a dictionary is usually faster than without one.

However, as the kinds of information captured in dictionaries grows, as described in Section 3.2.2, dictionaries may come to include instructions that significantly influence the speed of the compressor. For example, dictionaries might specify a particularly laborious transformation to be performed on the input. Or they might specify internal compression parameters, which might instruct the compressor to do huge amounts of work during compression.

If dictionary-based compressions systems evolve to include these sorts of features, care should be taken to avoid allowing dictionaries from untrusted sources to influence compression behavior or parameters. Note: this is not a concern for existing dictionaries.

Analogously, care should be taken to avoid allowing dictionaries to influence decompression performance.

##### 4.7.1.3.2. Generating Dictionaries

Training a dictionary, depending on the methodology, can be a very expensive computation (building an optimal dictionary is NP-hard). Designers of protocols that involve creating new dictionaries on the fly should constrain either or both of (1) who can cause a participant to train a new dictionary and (2) the computational cost of training a new dictionary (by selecting a fast algorithm or by limiting the amount of data over which the algorithm is run).

#### 4.7.2. Targets

In addition to the immediate compressing and decompressing agents, the mechanisms surrounding dictionary-based compression may allow for the targeting of other agents.

#### 4.7.2.1. An Intermediary

Insofar as intermediaries in internet protocols are often responsible for handling a much higher volume of traffic in a much lighter-weight way than protocol endpoints, any additional per-message or per-connection burden has the potential to significantly increase the workload of the intermediary. Retrieving, caching, and processing dictionaries, especially when the set of dictionaries is unbounded, is potentially untenable for intermediaries of that type.

#### 4.7.2.2. A Third Party

The mechanisms surrounding dictionary-based compression potentially also enable attacks against third parties, including parties with whom the attacker cannot exchange messages directly.

If a recipient can be induced to relay messages to a third-party, or to generate new messages directed at a third-party, a third party can become the effective recipient of dictionary-compressed traffic. If the dictionaries used to compress these messages are hard or slow to load (or even non-existent), the work of handling these messages could be significant. This is especially a risk when decompression of the message is required before it can be evaluated against an access-control policy or otherwise distinguished from legitimate traffic.

Protocol designers should therefore consider carefully the risks of using dictionary-based compression on (the parts of) messages that are used for authentication.

Another possible attack, when dictionaries are distributed dynamically, arises from the ability for compressed messages to trigger the retrieval of a dictionary from a third party. This is especially a risk when the source for a dictionary can be arbitrarily specified (as, for example, a URL).

These approaches potentially allow an attacker to amplify their efforts and turn their attack into a distributed one.

Protocol designers should consider how the source for the retrieval of a dictionary is derived, who can influence that derivation, and whether it should be constrained to preclude nominating a third party.

Protocol designers and implementors who relay messages should also consider whether those messages should be relayed compressed with the same dictionary, or whether dictionary selection and negotiation should occur for each hop in the path of a message.

#### 4.8. Generating Dictionaries

This section discusses the potential for inadvertent leakage of private information in the creation of dictionaries.

As described in Section 3.3.1, dictionaries are commonly generated by an algorithm run over a corpus sampled from the application's traffic. For systems which wish to publish dictionaries publicly (or, at any rate, with less strict access controls than the traffic on which they are trained), it is important to prevent the leakage of private information in the creation of dictionaries.

The output of this training process, the dictionary, as described in Section 3.2, may be composed of several different kinds of data. Some of these pieces, like statistical summaries around symbol frequencies, are unlikely to represent vectors for leaking useful information about the corpus they were trained on. Other components, however, directly represent substrings found in the input corpus.

Protocol designers, implementors, and participants that construct their own dictionaries should take care to do so in a way that does not reproduce private data in the produced dictionaries' contents.

##### 4.8.1. Handling Samples

Since dictionaries are generally produced from a collection of sample data, implementing a dictionary training capability may require storing or otherwise handling message traffic in ways it would otherwise not. This in itself can create an attack surface, for example if secrets that would normally exist only in transit or in memory are persisted or passed to other systems.

Care should be taken by implementors to protect the security of messages that are selected as samples for future use in dictionary training. Protections should be implemented both at rest and in transit, including retention limits, so as to limit the window of compromise.

##### 4.8.2. Tagging Mitigations

One strategy for ensuring that private data does not appear in dictionaries is to avoid presenting private data to the training algorithm at all. This sanitization of the training samples can be accomplished either by removing just the specific parts of samples that are private or by entirely removing samples that contain any private data in them.



This discrimination of private and public content can rely on being able to identify private information on sight (e.g., [CLOUDFLARE-NO-COMPRESS]).

Alternatively, the trainer can rely on explicit signals, provided alongside the messages, to perform that discrimination.

#### 4.8.3. Probabilistic Mitigations

Another strategy relies on a statistical approach for the identification and removal of private information.

In building the dictionary's contents, the goal of the dictionary training algorithm is to collect the set of strings that most effectively improve the compression ratio of messages in the corpus. This goal is best served by including strings that appear frequently in the sample corpus and rejecting strings that appear rarely.

In a loose way, it is reasonable to expect that commonly occurring substrings are less private, and rarely occurring substrings may be more private. So the dictionary trainer's interests are broadly aligned with this goal of not including private information in the dictionary.

While existing public dictionary training algorithms largely do not include specific protections or offer hard guarantees to prevent the inclusion of private data in their output, there is ongoing research in this area. Future algorithms may be able to provide confidence that private data (that is not somehow overrepresented in the training corpus) will be filtered out of the produced dictionary.

#### 4.9. Complexity

Complexity is ever the enemy of security. It is unavoidably the case that dictionary-based compression is more complicated than stateless compression.

#### 5. Conclusions

This document attempts to analyze risks and responses at the intersection of several widely varying factors--the protocol, the environment, the threat model--and its conclusions are necessarily situational.

From that space of configurations, some broad conclusions can nonetheless be drawn. Much of the complexity and risk in implementing dictionary-based compression comes from its surrounding apparatus: creating dictionaries, handling them, distributing them,

storing them, identifying them, and so on. A significant distinction can therefore be drawn between systems that have to grapple with those challenges versus those that don't.

[TODO]

## 6. IANA Considerations

This document includes no actions for IANA.

[RFC Editor: Please remove this section before publication.]

## 7. Security Considerations

This document enumerates known security considerations about a space that is under development. The list of issues discussed above may not be exhaustive, but it is hopefully complete enough to aid in the design and implementation of future systems and protocols.

## 8. References

### 8.1. Normative References

- [BROTLI] Alakuijala, J. and Z. Szabadka, "Brotli Compressed Data Format", RFC 7932, DOI 10.17487/RFC7932, July 2016, <<https://www.rfc-editor.org/info/rfc7932>>.
- [DEFLATE] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <<https://www.rfc-editor.org/info/rfc1951>>.
- [ZSTD] Collet, Y. and M. Kucherawy, Ed., "Zstandard Compression and the application/zstd Media Type", RFC 8478, DOI 10.17487/RFC8478, October 2018, <<https://www.rfc-editor.org/info/rfc8478>>.

### 8.2. Other Examples of Dictionary-Like Compression

- [HPACK] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <<https://www.rfc-editor.org/info/rfc7541>>.
- [HTTP-DELTA-ENCODING] Mogul, J., Krishnamurthy, B., Douglass, F., Feldmann, A., Goland, Y., van Hoff, A., and D. Hellerstein, "Delta encoding in HTTP", RFC 3229, DOI 10.17487/RFC3229, January 2002, <<https://www.rfc-editor.org/info/rfc3229>>.

- [I-D.ietf-quic-qpack]  
Krasic, C., Bishop, M., and A. Frindell, Ed., "QPACK: Header Compression for HTTP/3", draft-ietf-quic-qpack-10 (work in progress), 2019.
- [I-D.lee-sdch-spec]  
Butler, J., Lee, W., McQuade, B., and K. Mixter, "A Proposal for Shared Dictionary Compression over HTTP", draft-lee-sdch-spec-00 (work in progress), October 2016.
- [I-D.reschke-http-oob-encoding]  
Reschke, J. and S. Loreto, "'Out-Of-Band' Content Coding for HTTP", draft-reschke-http-oob-encoding-12 (work in progress), 2017.
- [I-D.vandevenne-shared-brotli-format]  
Alakuijala, J., Duong, T., Kliuchnikov, E., Obryk, R., Szabadka, Z., and L. Vandevenne, Ed., "Shared Brotli Compressed Data Format", draft-vandevenne-shared-brotli-format-04 (work in progress), August 2019.
- [I-D.vkrasnov-h2-compression-dictionaries]  
Krasnov, V. and Y. Weiss, "Compression Dictionaries for HTTP/2", draft-vkrasnov-h2-compression-dictionaries-03 (work in progress), 2018.

### 8.3. Informative References

- [BREACH] Prado, A., Harris, N., and Y. Gluck, "BREACH: SSL, Gone in 30 Seconds", 2013, <<https://breachattack.com/>>.
- [CLOUDFLARE-NO-COMPRESS]  
Loring, B., "A Solution to Compression Oracles on the Web", March 2018, <<https://blog.cloudflare.com/a-solution-to-compression-oracles-on-the-web/>>.
- [COOKIES] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [COVER] Liao, K., Petri, M., Moffat, A., and A. Wirth, "Effective Construction of Relative Lempel-Ziv Dictionaries", DOI 10.1145/2872427.2883042, 2016, <<https://doi.org/10.1145/2872427.2883042>>.
- [CRIME] Rizzo, J. and T. Duong, "Compression Ratio Info-leak Made Easy", 2012, <[https://www.ekoparty.org/archive/2012/CRIME\\_ekoparty2012.pdf](https://www.ekoparty.org/archive/2012/CRIME_ekoparty2012.pdf)>.

[ENCRYPT-THEN-AUTHENTICATE]

Krawczyk, H., "The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?)", 2001, <<https://iacr.org/archive/crypto2001/21390309.pdf>>.

[HEIST]

Vanhoef, M. and T. Van Goethem, "HEIST: HTTP Encrypted Information can be Stolen through TCP-windows", 2016, <[https://tom.vg/papers/heist\\_blackhat2016.pdf](https://tom.vg/papers/heist_blackhat2016.pdf)>.

[HTTP-CACHING]

Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/info/rfc7234>>.

[I-D.arkko-arch-internet-threat-model]

Arkko, J., "Changes in the Internet Threat Model", draft-arkko-arch-internet-threat-model-01 (work in progress), July 2019.

[I-D.draft-farrell-etm]

Farrell, S., "We're gonna need a bigger threat model", draft-farrell-etm-03 (work in progress), July 2019.

[I-D.draft-kucherawy-httpbis-dict-sec]

Kucherawy, M., "Security Considerations Regarding Compression Dictionaries", draft-kucherawy-httpbis-dict-sec-00 (work in progress), November 2018.

[I-D.pironti-tls-length-hiding]

Pironti, A. and N. Mavrogiannopoulos, "Length Hiding Padding for the Transport Layer Security Protocol", draft-pironti-tls-length-hiding-02 (work in progress), September 2013.

[LZ77]

Ziv, J. and A. Lempel, "A Universal Algorithm for Sequential Data Compression", DOI 10.1109/TIT.1977.1055714, May 1977, <<https://ieeexplore.ieee.org/document/1055714>>.

[LZ78]

Ziv, J. and A. Lempel, "Compression of individual sequences via variable-rate coding", DOI 10.1109/TIT.1978.1055934, September 1978, <<https://ieeexplore.ieee.org/document/1055934>>.

## [PERVASIVE-MONITORING]

Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.

## [PRIVACY]

Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.

## [RFC2360]

Scott, G., "Guide for Internet Standards Writers", BCP 22, RFC 2360, DOI 10.17487/RFC2360, June 1998, <<https://www.rfc-editor.org/info/rfc2360>>.

## [SECURITY-GUIDELINES]

Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.

## [SRI]

Akhawe, D., Braun, F., Marier, F., and J. Weinberger, "Subresource Integrity", March 2014, <<https://www.w3.org/TR/SRI/>>.

## [ZSTD-DICTS]

Collet, Y., Handte, W., and N. Terrell, "5 ways Facebook improved compression at scale with Zstandard", December 2018, <<https://code.fb.com/core-data/zstandard/>>.

## Appendix A. Acknowledgements

The author wishes to acknowledge the following for their help in writing and improving this document: Murray Kucherawy, Yann Collet, Nick Terrell, ... [TODO]

## Author's Address

W. Felix P. Handte  
Facebook, Inc.  
770 Broadway  
New York, NY 10003  
US

EMail: felixh@fb.com

HTTP  
Internet-Draft  
Obsoletes: 3205 (if approved)  
Intended status: Best Current Practice  
Expires: 28 February 2022

M. Nottingham  
27 August 2021

Building Protocols with HTTP  
draft-ietf-httpbis-bcp56bis-15

Abstract

Applications often use HTTP as a substrate to create HTTP-based APIs. This document specifies best practices for writing specifications that use HTTP to define new application protocols. It is written primarily to guide IETF efforts to define application protocols using HTTP for deployment on the Internet, but might be applicable in other situations.

This document obsoletes [RFC3205].

Note to Readers

\_RFC EDITOR: please remove this section before publication\_

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

Working Group information can be found at <http://httpwg.github.io/> (<http://httpwg.github.io/>); source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions/labels/bcp56bis> (<https://github.com/httpwg/http-extensions/labels/bcp56bis>).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 28 February 2022.

## Copyright Notice

Copyright (c) 2021 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 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 . . . . .	3
1.1. Notational Conventions . . . . .	5
2. Is HTTP Being Used? . . . . .	5
2.1. Non-HTTP Protocols . . . . .	5
3. What's Important About HTTP . . . . .	6
3.1. Generic Semantics . . . . .	6
3.2. Links . . . . .	7
3.3. Rich Functionality . . . . .	7
4. Best Practices for Specifying the Use of HTTP . . . . .	8
4.1. Specifying the Use of HTTP . . . . .	8
4.2. Specifying Server Behaviour . . . . .	9
4.3. Specifying Client Behaviour . . . . .	10
4.4. Specifying URLs . . . . .	11
4.4.1. Discovering an Application's URLs . . . . .	11
4.4.2. Considering URI Schemes . . . . .	12
4.4.3. Transport Ports . . . . .	13
4.5. Using HTTP Methods . . . . .	14
4.5.1. GET . . . . .	14
4.5.2. OPTIONS . . . . .	15
4.6. Using HTTP Status Codes . . . . .	16
4.6.1. Redirection . . . . .	17
4.7. Specifying HTTP Header Fields . . . . .	18
4.8. Defining Message Content . . . . .	20
4.9. Leveraging HTTP Caching . . . . .	20
4.9.1. Freshness . . . . .	20



4.9.2. Stale Responses . . . . .	21
4.9.3. Caching and Application Semantics . . . . .	21
4.9.4. Varying Content Based Upon the Request . . . . .	22
4.10. Handling Application State . . . . .	22
4.11. Making Multiple Requests . . . . .	22
4.12. Client Authentication . . . . .	23
4.13. Co-Existing with Web Browsing . . . . .	24
4.14. Maintaining Application Boundaries . . . . .	25
4.15. Using Server Push . . . . .	26
4.16. Allowing Versioning and Evolution . . . . .	27
5. IANA Considerations . . . . .	27
6. Security Considerations . . . . .	27
6.1. Privacy Considerations . . . . .	28
7. References . . . . .	29
7.1. Normative References . . . . .	29
7.2. Informative References . . . . .	30
Appendix A. Changes from RFC 3205 . . . . .	33
Author's Address . . . . .	33

## 1. Introduction

Applications other than Web browsing often use HTTP [HTTP] as a substrate, a practice sometimes referred to as creating "HTTP-based APIs", "REST APIs" or just "HTTP APIs". This is done for a variety of reasons, including:

- \* familiarity by implementers, specifiers, administrators, developers and users,
- \* availability of a variety of client, server and proxy implementations,
- \* ease of use,
- \* availability of Web browsers,
- \* reuse of existing mechanisms like authentication and encryption,
- \* presence of HTTP servers and clients in target deployments, and
- \* its ability to traverse firewalls.

These protocols are often ad hoc, intended for only deployment by one or a few servers and consumption by a limited set of clients. As a result, a body of practices and tools has arisen around defining HTTP-based APIs that favours these conditions.

However, when such an application has multiple, separate implementations, is deployed on multiple uncoordinated servers, and is consumed by diverse clients -- as is often the case for HTTP APIs defined by standards efforts -- tools and practices intended for limited deployment can become unsuitable.

This mismatch is largely because the API's clients and servers will implement and evolve at different paces, leading to a need for deployments with different features and versions to co-exist. As a result, the designers of HTTP-based APIs intended for such deployments need to more carefully consider how extensibility of the service will be handled and how different deployment requirements will be accommodated.

More generally, an application protocol using HTTP faces a number of design decisions, including:

- \* Should it define a new URI scheme? Use new ports?
- \* Should it use standard HTTP methods and status codes, or define new ones?
- \* How can the maximum value be extracted from the use of HTTP?
- \* How does it coexist with other uses of HTTP -- especially Web browsing?
- \* How can interoperability problems and "protocol dead ends" be avoided?

This document contains best current practices for the specification of such applications. Section 2 defines when it applies; Section 3 surveys the properties of HTTP that are important to preserve, and Section 4 conveys best practices for specifying them.

It is written primarily to guide IETF efforts to define application protocols using HTTP for deployment on the Internet, but might be applicable in other situations. Note that the requirements herein do not necessarily apply to the development of generic HTTP extensions.

This document obsoletes [RFC3205], to reflect experience and developments regarding HTTP in the intervening time.

### 1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2. Is HTTP Being Used?

Different applications have different goals when using HTTP. The recommendations in this document apply when a specification defines an application that:

- \* uses the transport port 80 or 443, or
- \* uses the URI scheme "http" or "https", or
- \* uses an ALPN protocol ID [RFC7301] that generically identifies HTTP (e.g., "http/1.1", "h2", "h3"), or
- \* makes registrations in or overall modifications to the IANA registries defined for HTTP.

Additionally, when a specification is using HTTP, all of the requirements of the HTTP protocol suite are in force (in particular, [HTTP], but also other specifications such as the specific version of HTTP in use, and any extensions in use).

Note that this document is intended to apply to applications, not generic extensions to HTTP. Furthermore, while it is intended for IETF-specified applications, other standards organisations are encouraged to adhere to its requirements.

### 2.1. Non-HTTP Protocols

An application can rely upon HTTP without meeting the criteria for using it defined above. For example, an application might wish to avoid re-specifying parts of the message format, but change other aspects of the protocol's operation; or, it might want to use application-specific methods.

Doing so brings more freedom to modify protocol operations, but loses at least a portion of the benefits outlined in Section 3, as most HTTP implementations won't be easily adaptable to these changes, and the benefit of mindshare will be lost.

Such specifications MUST NOT use HTTP's URI schemes, transport ports, ALPN protocol IDs or IANA registries; rather, they are encouraged to establish their own.

### 3. What's Important About HTTP

This section examines the characteristics of HTTP that are important to consider when using HTTP to define an application protocol.

#### 3.1. Generic Semantics

Much of the value of HTTP is in its generic semantics -- that is, the protocol elements defined by HTTP are potentially applicable to every resource, not specific to a particular context. Application-specific semantics are best expressed in message content and in header fields, not status codes or methods (although the latter do have generic semantics that relate to application state).

This generic/application-specific split allows a HTTP message to be handled by common software (e.g., HTTP servers, intermediaries, client implementations, and caches) without understanding the specific application. It also allows people to leverage their knowledge of HTTP semantics without specialising them for a particular application.

Therefore, applications that use HTTP MUST NOT re-define, refine or overlay the semantics of generic protocol elements such as methods, status codes or existing header fields. Instead, they should focus their specifications on protocol elements that are specific to that application; namely their HTTP resources.

When writing a specification, it's often tempting to specify exactly how HTTP is to be implemented, supported and used. However, this can easily lead to an unintended profile of HTTP's behaviour. For example, it's common to see specifications with language like this:

A 'POST' request MUST result in a '201 Created' response.

This forms an expectation in the client that the response will always be "201 Created", when in fact there are a number of reasons why the status code might differ in a real deployment; for example, there might be a proxy that requires authentication, or a server-side error, or a redirection. If the client does not anticipate this, the application's deployment is brittle.

See Section 4.2 for more details.

### 3.2. Links

Another common practice is assuming that the HTTP server's name space (or a portion thereof) is exclusively for the use of a single application. This effectively overlays special, application-specific semantics onto that space, precludes other applications from using it.

As explained in [RFC8820], such "squatting" on a part of the URL space by a standard usurps the server's authority over its own resources, can cause deployment issues, and is therefore bad practice in standards.

Instead of statically defining URI components like paths, it is RECOMMENDED that applications using HTTP define and use links [WEB-LINKING] to allow flexibility in deployment.

Using runtime links in this fashion has a number of other benefits -- especially when an application is to have multiple implementations and/or deployments (as is often the case for those that are standardised).

For example, navigating with a link allows a request to be routed to a different server without the overhead of a redirection, thereby supporting deployment across machines well.

It also becomes possible to "mix and match" different applications on the same server, and offers a natural mechanism for extensibility, versioning and capability management, since the document containing the links can also contain information about their targets.

Using links also offers a form of cache invalidation that's seen on the Web; when a resource's state changes, the application can change its link to it so that a fresh copy is always fetched.

### 3.3. Rich Functionality

HTTP offers a number of features to applications, such as:

- \* Message framing
- \* Multiplexing (in HTTP/2 [HTTP2] and HTTP/3 [HTTP3])
- \* Integration with TLS
- \* Support for intermediaries (proxies, gateways, Content Delivery Networks)

- \* Client authentication
- \* Content negotiation for format, language, and other features
- \* Caching for server scalability, latency and bandwidth reduction, and reliability
- \* Granularity of access control (through use of a rich space of URLs)
- \* Partial content to selectively request part of a response
- \* The ability to interact with the application easily using a Web browser

Applications that use HTTP are encouraged to utilise the various features that the protocol offers, so that their users receive the maximum benefit from it, and to allow it to be deployed in a variety of situations. This document does not require specific features to be used, since the appropriate design tradeoffs are highly specific to a given situation. However, following the practices in Section 4 is a good starting point.

#### 4. Best Practices for Specifying the Use of HTTP

This section contains best practices for specifying the use of HTTP by applications, including practices for specific HTTP protocol elements.

##### 4.1. Specifying the Use of HTTP

Specifications should use [HTTP] as the primary reference for HTTP; it is not necessary to reference all of the specifications in the HTTP suite unless there are specific reasons to do so (e.g., a particular feature is called out).

Because HTTP is a hop-by-hop protocol, a connection can be handled by implementations that are not controlled by the application; for example, proxies, CDNs, firewalls and so on. Requiring a particular version of HTTP makes it difficult to use in these situations, and harms interoperability. Therefore, it is NOT RECOMMENDED that applications using HTTP specify a minimum version of HTTP to be used.

However, if an application's deployment would benefit from the use of a particular version of HTTP (for example, HTTP/2's multiplexing), this ought be noted.

Applications using HTTP MUST NOT specify a maximum version, to preserve the protocol's ability to evolve.

When specifying examples of protocol interactions, applications should document both the request and response messages, with complete header sections, preferably in HTTP/1.1 format [HTTP11]. For example:

```
GET /thing HTTP/1.1
Host: example.com
Accept: application/things+json
User-Agent: Foo/1.0

HTTP/1.1 200 OK
Content-Type: application/things+json
Content-Length: 500
Server: Bar/2.2
```

[content here]

#### 4.2. Specifying Server Behaviour

The server-side behaviours of an application are most effectively specified by defining the following protocol elements:

- \* Media types [RFC6838], often based upon a format convention such as JSON [JSON],
- \* HTTP header fields, as per Section 4.7, and
- \* The behaviour of resources, as identified by link relations [WEB-LINKING].

An application can define its operation by composing these protocol elements to define a set of resources that are identified by link relations and that implement specified behaviours, including:

- \* retrieval of their state using GET, in one or more formats identified by media type;
- \* resource creation or update using POST or PUT, with an appropriately identified request content format;
- \* data processing using POST and identified request and response content format(s); and
- \* Resource deletion using DELETE.

For example, an application might specify:

Resources linked to with the "example-widget" link relation type are Widgets. The state of a Widget can be fetched in the "application/example-widget+json" format, and can be updated by PUT to the same link. Widget resources can be deleted.

The "Example-Count" response header field on Widget representations indicates how many Widgets are held by the sender.

The "application/example-widget+json" format is a JSON [RFC8259] format representing the state of a Widget. It contains links to related information in the link indicated by the Link header field value with the "example-other-info" link relation type.

Applications can also specify the use of URI Templates [URI-TEMPLATE] to allow clients to generate URLs based upon runtime data.

#### 4.3. Specifying Client Behaviour

An application's expectations for client behaviour ought to be closely aligned with those of Web browsers, to avoid interoperability issues when they are used.

One way to do this is to define it in terms of [FETCH], since that is the abstraction that browsers use for HTTP.

Some client behaviours (e.g., automatic redirect handling) and extensions (e.g., Cookies) are not required by HTTP, but nevertheless have become very common. If their use is not explicitly specified by applications using HTTP, there may be confusion and interoperability problems. In particular:

- \* Redirect handling - Applications need to specify how redirects are expected to be handled; see Section 4.6.1.
- \* Cookies - Applications using HTTP should explicitly reference the Cookie specification [COOKIES] if they are required.
- \* Certificates - Applications using HTTP should specify that TLS certificates are to be checked according to Section 4.3.4 of [HTTP] when HTTPS is used.



Applications using HTTP should not statically require HTTP features that are usually negotiated to be supported by clients. For example, requiring that clients support responses with a certain content-coding ([HTTP], Section 8.4.1) instead of negotiating for it ([HTTP], Section 12.5.3) means that otherwise conformant clients cannot interoperate with the application. Applications can encourage the implementation of such features, though.

#### 4.4. Specifying URLs

In HTTP, the resources that clients interact with are identified with URLs [URL]. As [RFC8820] explains, parts of the URL are designed to be under the control of the owner (also known as the "authority") of that server, to give them the flexibility in deployment.

This means that in most cases, specifications for applications that use HTTP won't contain fixed application URLs or paths; while it is common practice for a specification of a single-deployment API to specify the path prefix `"/app/v1"` (for example), doing so in an IETF specification is inappropriate.

Therefore, the specification writer needs some mechanism to allow clients to discover an application's URLs. Additionally, they need to specify what URL scheme(s) the application should be used with, and whether to use a dedicated port, or reuse HTTP's port(s).

##### 4.4.1. Discovering an Application's URLs

Generally, a client will begin interacting with a given application server by requesting an initial document that contains information about that particular deployment, potentially including links to other relevant resources. Doing so assures that the deployment is as flexible as possible (potentially spanning multiple servers), allows evolution, and also gives the application the opportunity to tailor the 'discovery document' to the client.

There are a few common patterns for discovering that initial URL.

The most straightforward mechanism for URL discovery is to configure the client with (or otherwise convey to it) a full URL. This might be done in a configuration document, or through another discovery mechanism.

However, if the client only knows the server's hostname and the identity of the application, there needs to be some way to derive the initial URL from that information.

An application cannot define a fixed prefix for its URL paths; see [RFC8820]. Instead, a specification for such an application can use one of the following strategies:

- \* Register a Well-Known URI [WELL-KNOWN-URI] as an entry point for that application. This provides a fixed path on every potential server that will not collide with other applications.
- \* Enable the server authority to convey a URI Template [URI-TEMPLATE] or similar mechanism for generating a URL for an entry point. For example, this might be done in a configuration document or other artefact.

Once the discovery document is located, it can be fetched, cached for later reuse (if allowed by its metadata), and used to locate other resources that are relevant to the application, using full URIs or URL Templates.

In some cases, an application may not wish to use such a discovery document; for example, when communication is very brief, or when the latency concerns of doing so precludes the use of a discovery document. These situations can be addressed by placing all of the application's resources under a well-known location.

#### 4.4.2. Considering URI Schemes

Applications that use HTTP will typically employ the "http" and/or "https" URI schemes. "https" is RECOMMENDED to provide authentication, integrity and confidentiality, as well as mitigate pervasive monitoring attacks [RFC7258].

However, application-specific schemes can also be defined. When defining an URI scheme for an application using HTTP, there are a number of tradeoffs and caveats to keep in mind:

- \* Unmodified Web browsers will not support the new scheme. While it is possible to register new URI schemes with Web browsers (e.g. `registerProtocolHandler()` in [HTML], as well as several proprietary approaches), support for these mechanisms is not shared by all browsers, and their capabilities vary.
- \* Existing non-browser clients, intermediaries, servers and associated software will not recognise the new scheme. For example, a client library might fail to dispatch the request; a cache might refuse to store the response, and a proxy might fail to forward the request.

- \* Because URLs occur in HTTP artefacts commonly, often being generated automatically (e.g., in the "Location" response header field), it can be difficult to assure that the new scheme is used consistently.
- \* The resources identified by the new scheme will still be available using "http" and/or "https" URLs. Those URLs can "leak" into use, which can present security and operability issues. For example, using a new scheme to assure that requests don't get sent to a "normal" Web site is likely to fail.
- \* Features that rely upon the URL's origin [RFC6454], such as the Web's same-origin policy, will be impacted by a change of scheme.
- \* HTTP-specific features such as cookies [COOKIES], authentication [HTTP], caching [HTTP-CACHING], HSTS [RFC6797], and CORS [FETCH] might or might not work correctly, depending on how they are defined and implemented. Generally, they are designed and implemented with an assumption that the URL will always be "http" or "https".
- \* Web features that require a secure context [SECCTXT] will likely treat a new scheme as insecure.

See [RFC7595] for more information about minting new URI schemes.

#### 4.4.3. Transport Ports

Applications can use the applicable default port (80 for HTTP, 443 for HTTPS), or they can be deployed upon other ports. This decision can be made at deployment time, or might be encouraged by the application's specification (e.g., by registering a port for that application).

If a non-default port is used, it needs to be reflected in the authority of all URLs for that resource; the only mechanism for changing a default port is changing the URI scheme (see Section 4.4.2).

Using a port other than the default has privacy implications (i.e., the protocol can now be distinguished from other traffic), as well as operability concerns (as some networks might block or otherwise interfere with it). Privacy implications (including those stemming from this distinguishability) should be documented in Security Considerations.

See [RFC7605] for further guidance.

#### 4.5. Using HTTP Methods

Applications that use HTTP MUST confine themselves to using registered HTTP methods such as GET, POST, PUT, DELETE, and PATCH.

New HTTP methods are rare; they are required to be registered in the HTTP Method Registry with IETF Review (see [HTTP]), and are also required to be generic. That means that they need to be potentially applicable to all resources, not just those of one application.

While historically some applications (e.g., [RFC4791]) have defined non-generic methods, [HTTP] now forbids this.

When authors believe that a new method is required, they are encouraged to engage with the HTTP community early (e.g., on the `ietf-http-wg@w3.org` mailing list), and document their proposal as a separate HTTP extension, rather than as part of an application's specification.

##### 4.5.1. GET

GET is the most common and useful HTTP method; its retrieval semantics allow caching, side-effect free linking and underlies many of the benefits of using HTTP.

Queries can be performed with GET, often using the query component of the URL; this is a familiar pattern from Web browsing, and the results can be cached, improving efficiency of an often expensive process. In some cases, however, GET might be unwieldy for expressing queries, because of the limited syntax of the URI; in particular, if binary data forms part of the query terms, it needs to be encoded to conform to URI syntax.

While this is not an issue for short queries, it can become one for larger query terms, or ones which need to sustain a high rate of requests. Additionally, some HTTP implementations limit the size of URLs they support -- although modern HTTP software has much more generous limits than previously (typically, considerably more than 8000 octets, as required by [HTTP]).

In these cases, an application using HTTP might consider using POST to express queries in the request's content; doing so avoids encoding overhead and URL length limits in implementations. However, in doing so it should be noted that the benefits of GET such as caching and linking to query results are lost. Therefore, applications using HTTP that feel a need to allow POST queries ought to consider allowing both methods.

Processing of GET requests should not change application state or have other side effects that might be significant to the client, since implementations can and do retry HTTP GET requests that fail, and some GET requests protected by TLS Early Data might be vulnerable to replay attacks (see [RFC8470]). Note that this does not include logging and similar functions; see [HTTP], Section 9.2.1.

Finally, note that while the generic HTTP syntax allows a GET request message to contain content, the purpose is to allow message parsers to be generic; as per [HTTP], Section 9.3.1, content on a GET is not recommended, has no meaning, and will be either ignored or rejected by generic HTTP software (such as intermediaries, caches, servers, and client libraries).

#### 4.5.2. OPTIONS

The OPTIONS method was defined for metadata retrieval, and is used both by WebDAV [RFC4918] and CORS [FETCH]. Because HTTP-based APIs often need to retrieve metadata about resources, it is often considered for their use.

However, OPTIONS does have significant limitations:

- \* It isn't possible to link to the metadata with a simple URL, because OPTIONS is not the default method.
- \* OPTIONS responses are not cacheable, because HTTP caches operate on representations of the resource (i.e., GET and HEAD). If OPTIONS responses are cached separately, their interaction with HTTP cache expiry, secondary keys and other mechanisms needs to be considered.
- \* OPTIONS is "chatty" - always separating metadata out into a separate request increases the number of requests needed to interact with the application.
- \* Implementation support for OPTIONS is not universal; some servers do not expose the ability to respond to OPTIONS requests without significant effort.

Instead of OPTIONS, one of these alternative approaches might be more appropriate:

- \* For server-wide metadata, create a well-known URI [WELL-KNOWN-URI], or use an already existing one if appropriate (e.g., HostMeta [RFC6415]).

- \* For metadata about a specific resource, create a separate resource and link to it using a Link response header field or a link serialised into the response's content. See [WEB-LINKING]. Note that the Link header field is available on HEAD responses, which is useful if the client wants to discover a resource's capabilities before they interact with it.

#### 4.6. Using HTTP Status Codes

HTTP status codes convey semantics both for the benefit of generic HTTP components -- such as caches, intermediaries, and clients -- and applications themselves. However, applications can encounter a number of pitfalls in their use.

First, status codes are often generated by components other than the application itself. This can happen, for example, when network errors are encountered, a captive portal, proxy or Content Delivery Network is present, when a server is overloaded, or it thinks it is under attack. They can even be generated by generic client software when certain error conditions are encountered. As a result, if an application assigns specific semantics to one of these status codes, a client can be misled about its state, because the status code was generated by a generic component, not the application itself.

Furthermore, mapping application errors to individual HTTP status codes one-to-one often leads to a situation where the finite space of applicable HTTP status codes is exhausted. This, in turn, leads to a number of bad practices -- including minting new, application-specific status codes, or using existing status codes even though the link between their semantics and the application's is tenuous at best.

Instead, applications using HTTP should define their errors to use the most applicable status code, making generous use of the general status codes (200, 400 and 500) when in doubt. Importantly, they should not specify a one-to-one relationship between status codes and application errors, thereby avoiding the exhaustion issue outlined above.

To distinguish between multiple error conditions that are mapped to the same status code, and to avoid the misattribution issue outlined above, applications using HTTP should convey finer-grained error information in the response's message content and/or header fields. [PROBLEM-DETAILS] provides one way to do so.

Because the set of registered HTTP status codes can expand, applications using HTTP should explicitly point out that clients ought to be able to handle all applicable status codes gracefully

(i.e., falling back to the generic "n00" semantics of a given status code; e.g., "499" can be safely handled as "400" by clients that don't recognise it). This is preferable to creating a "laundry list" of potential status codes, since such a list won't be complete in the foreseeable future.

Applications using HTTP MUST NOT re-specify the semantics of HTTP status codes, even if it is only by copying their definition. It is NOT RECOMMENDED they require specific reason phrases to be used; the reason phrase has no function in HTTP, is not guaranteed to be preserved by implementations, and is not carried at all in the HTTP/2 HTTP2 message format.

Applications MUST only use registered HTTP status codes. As with methods, new HTTP status codes are rare, and required (by [HTTP]) to be registered with IETF Review. Similarly, HTTP status codes are generic; they are required (by [HTTP]) to be potentially applicable to all resources, not just to those of one application.

When authors believe that a new status code is required, they are encouraged to engage with the HTTP community early (e.g., on the [ietf-http-wg@w3.org](mailto:ietf-http-wg@w3.org) mailing list), and document their proposal as a separate HTTP extension, rather than as part of an application's specification.

#### 4.6.1. Redirection

The 3xx series of status codes specified in Section 15.4 of [HTTP] direct the user agent to another resource to satisfy the request. The most common of these are 301, 302, 307 and 308, all of which use the Location response header field to indicate where the client should resend the request.

There are two ways that the members of this group of status codes differ:

- \* Whether they are permanent or temporary. Permanent redirects can be used to update links stored in the client (e.g., bookmarks), whereas temporary ones cannot. Note that this has no effect on HTTP caching; it is completely separate.
- \* Whether they allow the redirected request to change the request method from POST to GET. Web browsers generally do change POST to GET for 301 and 302; therefore, 308 and 307 were created to allow redirection without changing the method.

This table summarises their relationships:

	Permanent	Temporary
Allows changing the request method from POST to GET	301	302
Does not allow changing the request method	308	307

Table 1

The 303 See Other status code can be used to inform the client that the result of an operation is available at a different location using GET.

As noted in [HTTP], a user agent is allowed to automatically follow a 3xx redirect that has a Location response header field, even if they don't understand the semantics of the specific status code. However, they aren't required to do so; therefore, if an application using HTTP desires redirects to be automatically followed, it needs to explicitly specify the circumstances when this is required.

Redirects can be cached (when appropriate cache directives are present), but beyond that they are not 'sticky' -- i.e., redirection of a URI will not result in the client assuming that similar URIs (e.g., with different query parameters) will also be redirected.

Applications using HTTP are encouraged to specify that 301 and 302 responses change the subsequent request method from POST (but no other method) to GET, to be compatible with browsers. Generally, when a redirected request is made, its header fields are copied from the original request's. However, they can be modified by various mechanisms; e.g., sent Authorization ([HTTP], Section 11) and Cookie ([COOKIES]) header fields will change if the origin (and sometimes path) of the request changes. An application using HTTP should specify if any request header fields that it defines need to be modified or removed upon a redirect; however, this behaviour cannot be relied upon, since a generic client (like a browser) will be unaware of such requirements.

#### 4.7. Specifying HTTP Header Fields

Applications often define new HTTP header fields. Typically, using HTTP header fields is appropriate in a few different situations:

- \* The field is useful to intermediaries (who often wish to avoid parsing message content), and/or



- \* The field is useful to generic HTTP software (e.g., clients, servers), and/or
- \* It is not possible to include their values in the message content (usually because a format does not allow it).

When the conditions above are not met, it is usually better to convey application-specific information in other places; e.g., the message content or the URL query string.

New header fields **MUST** be registered, as per Section 16.3 of [HTTP].

See Section 16.3.2 of [HTTP] for guidelines to consider when minting new header fields. [STRUCTURED-FIELDS] provides a common structure for new header fields, and avoids many issues in their parsing and handling; it is **RECOMMENDED** that new header fields use it.

It is **RECOMMENDED** that header field names be short (even when field compression is used, there is an overhead) but appropriately specific. In particular, if a header field is specific to an application, an identifier for that application can form a prefix to the header field name, separated by a "-".

For example, if the "example" application needs to create three header fields, they might be called "example-foo", "example-bar" and "example-baz". Note that the primary motivation here is to avoid consuming more generic field names, not to reserve a portion of the namespace for the application; see [RFC6648] for related considerations.

The semantics of existing HTTP header fields **MUST NOT** be re-defined without updating their registration or defining an extension to them (if allowed). For example, an application using HTTP cannot specify that the "Location" header field has a special meaning in a certain context.

See Section 4.9 for the interaction between header fields and HTTP caching; in particular, request header fields that are used to "select" a response have impact there, and need to be carefully considered.

See Section 4.10 for considerations regarding header fields that carry application state (e.g., Cookie).

#### 4.8. Defining Message Content

Common syntactic conventions for message contents include JSON [JSON], XML [XML], and CBOR [RFC8949]. Best practices for their use are out of scope for this document.

Applications should register distinct media types for each format they define; this makes it possible to identify them unambiguously and negotiate for their use. See [RFC6838] for more information.

#### 4.9. Leveraging HTTP Caching

HTTP caching [HTTP-CACHING] is one of the primary benefits of using HTTP for applications; it provides scalability, reduces latency and improves reliability. Furthermore, HTTP caches are readily available in browsers and other clients, networks as forward and reverse proxies, Content Delivery Networks and as part of server software.

Even when an application using HTTP isn't designed to take advantage of caching, it needs to consider how caches will handle its responses, to preserve correct behaviour when one is interposed (whether in the network, server, client, or intervening infrastructure).

##### 4.9.1. Freshness

Assigning even a short freshness lifetime ([HTTP-CACHING], Section 4.2) -- e.g., 5 seconds -- allows a response to be reused to satisfy multiple clients, and/or a single client making the same request repeatedly. In general, if it is safe to reuse something, consider assigning a freshness lifetime.

The most common method for specifying freshness is the max-age response directive ([HTTP-CACHING], Section 5.2.2.1). The Expires header field ([HTTP-CACHING], Section 5.3) can also be used, but it is not necessary; all modern cache implementations support Cache-Control, and specifying freshness as a delta is usually more convenient and less error-prone.

It is not necessary to add the "public" response directive ([HTTP-CACHING], Section 5.2.2.9) to cache most responses; it is only necessary when it's desirable to store an authenticated response, or when the status code isn't understood by the cache and there isn't explicit freshness information available.

In some situations, responses without explicit cache freshness directives will be stored and served using a heuristic freshness lifetime; see [HTTP-CACHING], Section 4.2.2. As the heuristic is not

under control of the application, it is generally preferable to set an explicit freshness lifetime, or make the response explicitly uncacheable.

If caching of a response is not desired, the appropriate response directive is "Cache-Control: no-store". Other directives are not necessary, and no-store only need be sent in situations where the response might be cached; see [HTTP-CACHING], Section 3. Note that "Cache-Control: no-cache" allows a response to be stored, just not reused by a cache without validation; it does not prevent caching (despite its name).

For example, this response cannot be stored or reused by a cache:

```
HTTP/1.1 200 OK
Content-Type: application/example+xml
Cache-Control: no-store
```

[content]

#### 4.9.2. Stale Responses

Authors should understand that stale responses (e.g., with "Cache-Control: max-age=0") can be reused by caches when disconnected from the origin server; this can be useful for handling network issues.

If doing so is not suitable for a given response, the origin should use "Cache-Control: must-revalidate". See Section 4.2.4 of [HTTP-CACHING], and also [RFC5861] for additional controls over stale content.

Stale responses can be refreshed by assigning a validator, saving both transfer bandwidth and latency for large responses; see Section 13 of [HTTP].

#### 4.9.3. Caching and Application Semantics

When an application has a need to express a lifetime that's separate from the freshness lifetime, this should be conveyed separately, either in the response's content or in a separate header field. When this happens, the relationship between HTTP caching and that lifetime needs to be carefully considered, since the response will be used as long as it is considered fresh.

In particular, application authors need to consider how responses that are not freshly obtained from the origin server should be handled; if they have a concept like a validity period, this will need to be calculated considering the age of the response (see [HTTP-CACHING], Section 4.2.3).

One way to address this is to explicitly specify that responses need to be fresh upon use.

#### 4.9.4. Varying Content Based Upon the Request

If an application uses a request header field to change the response's header fields or content, authors should point out that this has implications for caching; in general, such resources need to either make their responses uncacheable (e.g., with the "no-store" cache-control directive defined in [HTTP-CACHING], Section 5.2.2.5) or send the Vary response header field ([HTTP], Section 12.5.5) on all responses from that resource (including the "default" response).

For example, this response:

```
HTTP/1.1 200 OK
Content-Type: application/example+xml
Cache-Control: max-age=60
ETag: "sa0f8wf20fs0f"
Vary: Accept-Encoding
```

[content]

can be stored for 60 seconds by both private and shared caches, can be revalidated with If-None-Match, and varies on the Accept-Encoding request header field.

#### 4.10. Handling Application State

Applications can use stateful cookies [COOKIES] to identify a client and/or store client-specific data to contextualise requests.

When used, it is important to carefully specify the scoping and use of cookies; if the application exposes sensitive data or capabilities (e.g., by acting as an ambient authority), exploits are possible. Mitigations include using a request-specific token to assure the intent of the client.

#### 4.11. Making Multiple Requests

Clients often need to send multiple requests to perform a task.

In HTTP/1 [HTTP11], parallel requests are most often supported by opening multiple connections. Application performance can be impacted when too many simultaneous connections are used, because connections' congestion control will not be coordinated. Furthermore, it can be difficult for applications to decide when to issue and which connection to use for a given request, further impacting performance.

HTTP/2 [HTTP2] and HTTP/3 [HTTP3] offer multiplexing to applications, removing the need to use multiple connections. However, application performance can still be significantly affected by how the server chooses to prioritize responses. Depending on the application, it might be best for the server to determine the priority of responses, or for the client to hint its priorities to the server (see, e.g., [HTTP-PRIORITY]).

In all versions of HTTP, requests are made independently -- you can't rely on the relative order of two requests to guarantee processing order. This is because they might be sent over a multiplexed protocol by an intermediary, sent to different origin servers, or the server might even perform processing in a different order. If two requests need strict ordering, the only reliable way to assure the outcome is to issue the second request when the final response to the first has begun.

Applications MUST NOT make assumptions about the relationship between separate requests on a single transport connection; doing so breaks many of the assumptions of HTTP as a stateless protocol, and will cause problems in interoperability, security, operability and evolution.

#### 4.12. Client Authentication

Applications can use HTTP authentication Section 11 of [HTTP] to identify clients. As per [RFC7617], the Basic authentication scheme is not suitable for protecting sensitive or valuable information unless the channel is secure (e.g., using the "HTTPS" URI scheme). Likewise, [RFC7616] requires the Digest authentication scheme to be used over a secure channel.

With HTTPS, clients might also be authenticated using certificates [RFC8446], but note that such authentication is intrinsically scoped to the underlying transport connection. As a result, a client has no way of knowing whether the authenticated status was used in preparing the response (though "Vary: \*" and/or "Cache-Control: private" can provide a partial indication), and the only way to obtain a specifically unauthenticated response is to open a new connection.

When used, it is important to carefully specify the scoping and use of authentication; if the application exposes sensitive data or capabilities (e.g., by acting as an ambient authority; see Section 8.3 of [RFC6454]), exploits are possible. Mitigations include using a request-specific token to assure the intent of the client.

#### 4.13. Co-Existing with Web Browsing

Even if there is not an intent for an application to be used with a Web browser, its resources will remain available to browsers and other HTTP clients. This means that all such applications that use HTTP need to consider how browsers will interact with them, particularly regarding security.

For example, if an application's state can be changed using a POST request, a Web browser can easily be coaxed into cross-site request forgery (CSRF) from arbitrary Web sites.

Or, if an attacker gains control of content returned from the application's resources (for example, part of the request is reflected in the response, or the response contains external information that the attacker can change), they can inject code into the browser and access data and capabilities as if they were the origin -- a technique known as a cross-site scripting (XSS) attack.

This is only a small sample of the kinds of issues that applications using HTTP must consider. Generally, the best approach is to actually consider the application as a Web application, and to follow best practices for their secure development.

A complete enumeration of such practices is out of scope for this document, but some considerations include:

- \* Using an application-specific media type in the Content-Type header field, and requiring clients to fail if it is not used.
- \* Using X-Content-Type-Options: nosniff [FETCH] to assure that content under attacker control can't be coaxed into a form that is interpreted as active content by a Web browser.
- \* Using Content-Security-Policy [CSP] to constrain the capabilities of active content (i.e., that which can execute scripts, such as HTML [HTML] and PDF), thereby mitigating Cross-Site Scripting attacks.
- \* Using Referrer-Policy [REFERRER-POLICY] to prevent sensitive data in URLs from being leaked in the Referer request header field.

- \* Using the 'HttpOnly' flag on Cookies to assure that cookies are not exposed to browser scripting languages [COOKIES].
- \* Avoiding use of compression on any sensitive information (e.g., authentication tokens, passwords), as the scripting environment offered by Web browsers allows an attacker to repeatedly probe the compression space; if the attacker has access to the path of the communication, they can use this capability to recover that information.

Depending on how they are intended to be deployed, specifications for applications using HTTP might require the use of these mechanisms in specific ways, or might merely point them out in Security Considerations.

An example of a HTTP response from an application that does not intend for its content to be treated as active by browsers might look like this:

```
HTTP/1.1 200 OK
Content-Type: application/example+json
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'none'
Cache-Control: max-age=3600
Referrer-Policy: no-referrer
```

[content]

If an application has browser compatibility as a goal, client interaction ought to be defined in terms of [FETCH], since that is the abstraction that browsers use for HTTP; it enforces many of these best practices.

#### 4.14. Maintaining Application Boundaries

Because many HTTP capabilities are scoped to the origin [RFC6454], applications also need to consider how deployments might interact with other applications (including Web browsing) on the same origin.

For example, if Cookies [COOKIES] are used to carry application state, they will be sent with all requests to the origin by default (unless scoped by path), and the application might receive cookies from other applications on the origin. This can lead to security issues, as well as collision in cookie names.

One solution to these issues is to require a dedicated hostname for the application, so that it has a unique origin. However, it is often desirable to allow multiple applications to be deployed on a

single hostname; doing so provides the most deployment flexibility and enables them to be "mixed" together (See [RFC8820] for details). Therefore, applications using HTTP should strive to allow multiple applications on an origin.

To enable this, when specifying the use of Cookies, HTTP authentication realms [HTTP], or other origin-wide HTTP mechanisms, applications using HTTP should not mandate the use of a particular name, but instead let deployments configure them. Consideration should be given to scoping them to part of the origin, using their specified mechanisms for doing so.

Modern Web browsers constrain the ability of content from one origin to access resources from another, to avoid leaking private information. As a result, applications that wish to expose cross-origin data to browsers will need to implement the CORS protocol; see [FETCH].

#### 4.15. Using Server Push

HTTP/2 added the ability for servers to "push" request/response pairs to clients in [HTTP2], Section 8.4. While server push seems like a natural fit for many common application semantics (e.g., "fanout" and publish/subscribe), a few caveats should be noted:

- \* Server push is hop-by-hop; that is, it is not automatically forwarded by intermediaries. As a result, it might not work easily (or at all) with proxies, reverse proxies, and Content Delivery Networks.
- \* Server push can have negative performance impact on HTTP when used incorrectly; in particular, if there is contention with resources that have actually been requested by the client.
- \* Server push is implemented differently in different clients, especially regarding interaction with HTTP caching, and capabilities might vary.
- \* APIs for server push are currently unavailable in some implementations, and vary widely in others. In particular, there is no current browser API for it.
- \* Server push is not supported in HTTP/1.1 or HTTP/1.0.
- \* Server push does not form part of the "core" semantics of HTTP, and therefore might not be supported by future versions of the protocol.



Applications wishing to optimise cases where the client can perform work related to requests before the full response is available (e.g., fetching links for things likely to be contained within) might benefit from using the 103 (Early Hints) status code; see [RFC8297].

Applications using server push directly need to enforce the requirements regarding authority in [HTTP2], Section 8.4, to avoid cross-origin push attacks.

#### 4.16. Allowing Versioning and Evolution

It's often necessary to introduce new features into application protocols, and change existing ones.

In HTTP, backwards-incompatible changes can be made using mechanisms such as:

- \* Using a distinct link relation type [WEB-LINKING] to identify a URL for a resource that implements the new functionality.
- \* Using a distinct media type [RFC6838] to identify formats that enable the new functionality.
- \* Using a distinct HTTP header field to implement new functionality outside the message content.

#### 5. IANA Considerations

This document has no requirements for IANA.

#### 6. Security Considerations

Applications using HTTP are subject to the security considerations of HTTP itself and any extensions used; [HTTP], [HTTP-CACHING], and [WEB-LINKING] are often relevant, amongst others.

Section 4.4.2 recommends support for 'https' URLs, and discourages the use of 'http' URLs, to provide authentication, integrity and confidentiality, as well as mitigate pervasive monitoring attacks. Many applications using HTTP perform authentication and authorization with bearer tokens (e.g., in session cookies). If the transport is unencrypted, an attacker that can eavesdrop upon or modify HTTP communications can often escalate their privilege to perform operations on resources.

Section 4.9.3 highlights the potential for mismatch between HTTP caching and application-specific storage of responses or information therein.

Section 4.10 discusses the impact of using stateful mechanisms in the protocol as ambient authority, and suggests a mitigation.

Section 4.13 highlights the implications of Web browsers' capabilities on applications that use HTTP.

Section 4.14 discusses the issues that arise when applications are deployed on the same origin as Web sites (and other applications).

Section 4.15 highlights risks of using HTTP/2 server push in a manner other than specified.

Applications that use HTTP in a manner that involves modification of implementations -- for example, requiring support for a new URI scheme, or a non-standard method -- risk having those implementations "fork" from their parent HTTP implementations, with the possible result that they do not benefit from patches and other security improvements incorporated upstream.

#### 6.1. Privacy Considerations

HTTP clients can expose a variety of information to servers. Besides information that's explicitly sent as part of an application's operation (for example, names and other user-entered data), and "on the wire" (which is one of the reasons https is recommended in Section 4.4.2), other information can be gathered through less obvious means -- often by connecting activities of a user over time.

This includes session information, tracking the client through fingerprinting, and code execution.

Session information includes things like the IP address of the client, TLS session tickets, Cookies, ETags stored in the client's cache, and other stateful mechanisms. Applications are advised to avoid using session mechanisms unless they are unavoidable or necessary for operation, in which case these risks need to be documented. When they are used, implementations should be encouraged to allow clearing such state.

Fingerprinting uses unique aspects of a client's messages and behaviours to connect disparate requests and connections. For example, the User-Agent request header field conveys specific information about the implementation; the Accept-Language request header field conveys the users' preferred language. In combination, a number of these markers can be used to uniquely identify a client, impacting its control over its data. As a result, applications are advised to specify that clients should only emit the information they need to function in requests.

Finally, if an application exposes the ability to execute code, great care needs to be taken, since any ability to observe its environment can be used as an opportunity to both fingerprint the client and to obtain and manipulate private data (including session information). For example, access to high-resolution timers (even indirectly) can be used to profile the underlying hardware, creating a unique identifier for the system. Applications are advised to avoid allowing the use of mobile code where possible; when it cannot be avoided, the resulting system's security properties need be carefully scrutinised.

## 7. References

### 7.1. Normative References

- [HTTP] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-18, 18 August 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-18>>.
- [HTTP-CACHING] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Caching", Work in Progress, Internet-Draft, draft-ietf-httpbis-cache-18, 18 August 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cache-18>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/rfc/rfc6454>>.
- [RFC6648] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, DOI 10.17487/RFC6648, June 2012, <<https://www.rfc-editor.org/rfc/rfc6648>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/rfc/rfc6838>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC8820] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 8820, DOI 10.17487/RFC8820, June 2020, <<https://www.rfc-editor.org/rfc/rfc8820>>.
- [URL] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [WEB-LINKING]  
Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/rfc/rfc8288>>.
- [WELL-KNOWN-URI]  
Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <<https://www.rfc-editor.org/rfc/rfc8615>>.

## 7.2. Informative References

- [COOKIES] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/rfc/rfc6265>>.
- [CSP] West, M., "Content Security Policy Level 3", World Wide Web Consortium WD WD-CSP3-20160913, 13 September 2016, <<https://www.w3.org/TR/2016/WD-CSP3-20160913>>.
- [FETCH] WHATWG, "Fetch - Living Standard", n.d., <<https://fetch.spec.whatwg.org>>.
- [HTML] WHATWG, "HTML - Living Standard", n.d., <<https://html.spec.whatwg.org>>.
- [HTTP-PRIORITY]  
Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-priority-04, 11 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-priority-04>>.

- [HTTP11] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-18, 18 August 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-18>>.
- [HTTP2] Thomson, M. and C. Benfield, "Hypertext Transfer Protocol Version 2 (HTTP/2)", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2bis-03, 12 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2bis-03>>.
- [HTTP3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>>.
- [JSON] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/rfc/rfc8259>>.
- [PROBLEM-DETAILS] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <<https://www.rfc-editor.org/rfc/rfc7807>>.
- [REFERRER-POLICY] Eisinger, J. and E. Stark, "Referrer Policy", World Wide Web Consortium CR CR-referrer-policy-20170126, 26 January 2017, <<https://www.w3.org/TR/2017/CR-referrer-policy-20170126>>.
- [RFC3205] Moore, K., "On the use of HTTP as a Substrate", BCP 56, RFC 3205, DOI 10.17487/RFC3205, February 2002, <<https://www.rfc-editor.org/rfc/rfc3205>>.
- [RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, DOI 10.17487/RFC4791, March 2007, <<https://www.rfc-editor.org/rfc/rfc4791>>.
- [RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, DOI 10.17487/RFC4918, June 2007, <<https://www.rfc-editor.org/rfc/rfc4918>>.

- [RFC5861] Nottingham, M., "HTTP Cache-Control Extensions for Stale Content", RFC 5861, DOI 10.17487/RFC5861, May 2010, <<https://www.rfc-editor.org/rfc/rfc5861>>.
- [RFC6415] Hammer-Lahav, E., Ed. and B. Cook, "Web Host Metadata", RFC 6415, DOI 10.17487/RFC6415, October 2011, <<https://www.rfc-editor.org/rfc/rfc6415>>.
- [RFC6797] Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", RFC 6797, DOI 10.17487/RFC6797, November 2012, <<https://www.rfc-editor.org/rfc/rfc6797>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/rfc/rfc7258>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/rfc/rfc7301>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<https://www.rfc-editor.org/rfc/rfc7595>>.
- [RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <<https://www.rfc-editor.org/rfc/rfc7605>>.
- [RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <<https://www.rfc-editor.org/rfc/rfc7616>>.
- [RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <<https://www.rfc-editor.org/rfc/rfc7617>>.
- [RFC8297] Oku, K., "An HTTP Status Code for Indicating Hints", RFC 8297, DOI 10.17487/RFC8297, December 2017, <<https://www.rfc-editor.org/rfc/rfc8297>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/rfc/rfc8446>>.

- [RFC8470] Thomson, M., Nottingham, M., and W. Tarreau, "Using Early Data in HTTP", RFC 8470, DOI 10.17487/RFC8470, September 2018, <<https://www.rfc-editor.org/rfc/rfc8470>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/rfc/rfc8949>>.
- [SECCTXT] West, M., "Secure Contexts", World Wide Web Consortium CR CR-secure-contexts-20160915, 15 September 2016, <<https://www.w3.org/TR/2016/CR-secure-contexts-20160915>>.
- [STRUCTURED-FIELDS]  
Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/rfc/rfc8941>>.
- [URI-TEMPLATE]  
Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/rfc/rfc6570>>.
- [XML] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, 26 November 2008, <<https://www.w3.org/TR/2008/REC-xml-20081126>>.

#### Appendix A. Changes from RFC 3205

[RFC3205] captured the Best Current Practice in the early 2000's, based on the concerns facing protocol designers at the time. Use of HTTP has changed considerably since then, and as a result this document is substantially different. As a result, the changes are too numerous to list individually.

#### Author's Address

Mark Nottingham  
Pahran  
Australia

Email: [mnot@mnot.net](mailto:mnot@mnot.net)  
URI: <https://www.mnot.net/>

HTTP  
Internet-Draft  
Intended status: Standards Track  
Expires: 18 February 2022

M. Nottingham  
Fastly  
17 August 2021

The Cache-Status HTTP Response Header Field  
draft-ietf-httpbis-cache-header-10

Abstract

To aid debugging, HTTP caches often append header fields to a response explaining how they handled the request in an ad hoc manner. This specification defines a standard mechanism to do so that is aligned with HTTP's caching model.

Note to Readers

\_RFC EDITOR: please remove this section before publication\_

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

Working Group information can be found at <https://httpwg.org/> (<https://httpwg.org/>); source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions/labels/cache-header> (<https://github.com/httpwg/http-extensions/labels/cache-header>).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 18 February 2022.



## Copyright Notice

Copyright (c) 2021 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 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 . . . . .	2
1.1. Notational Conventions . . . . .	3
2. The Cache-Status HTTP Response Header Field . . . . .	3
2.1. The hit parameter . . . . .	4
2.2. The fwd parameter . . . . .	4
2.3. The fwd-status parameter . . . . .	5
2.4. The ttl parameter . . . . .	6
2.5. The stored parameter . . . . .	6
2.6. The collapsed parameter . . . . .	6
2.7. The key parameter . . . . .	6
2.8. The detail parameter . . . . .	6
3. Examples . . . . .	7
4. Defining New Cache-Status Parameters . . . . .	8
5. IANA Considerations . . . . .	8
6. Security Considerations . . . . .	9
7. References . . . . .	9
7.1. Normative References . . . . .	9
7.2. Informative References . . . . .	10
Author's Address . . . . .	10

## 1. Introduction

To aid debugging (both by humans and automated tools), HTTP caches often append header fields to a response explaining how they handled the request. Unfortunately, the semantics of these headers are often unclear, and both the semantics and syntax used vary between implementations.

This specification defines a new HTTP response header field, "Cache-Status" for this purpose, with standardized syntax and semantics.

### 1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses ABNF as defined in [RFC5234], with rules prefixed with "sf-" and the "key" rule as defined in [STRUCTURED-FIELDS]. It uses terminology from [HTTP] and [HTTP-CACHING].

## 2. The Cache-Status HTTP Response Header Field

The Cache-Status HTTP response header field indicates how caches have handled that response and its corresponding request. The syntax of this header field conforms to [STRUCTURED-FIELDS].

Its value is a List ([STRUCTURED-FIELDS], Section 3.1):

Cache-Status = sf-list

Each member of the list represents a cache that has handled the request. The first member of the list represents the cache closest to the origin server, and the last member of the list represents the cache closest to the user (possibly including the user agent's cache itself, if it appends a value).

Caches determine when it is appropriate to add the Cache-Status header field to a response. Some might add it to all responses, whereas others might only do so when specifically configured to, or when the request contains a header field that activates a debugging mode. See Section 6 for related security considerations.

An intermediary SHOULD NOT append a Cache-Status member to responses that it generates locally, even if that intermediary contains a cache, unless the generated response is based upon a stored response (e.g., 304 Not Modified and 206 Partial Content are both based upon a stored response). For example, a proxy generating a 400 response due to a malformed request will not add a Cache-Status value, because that response was generated by the proxy, not the origin server.

When adding a value to the Cache-Status header field, caches SHOULD preserve the existing field value, to allow debugging of the entire chain of caches handling the request.

Each list member identifies the cache that inserted it and this identifier MUST be a String or Token. Depending on the deployment, this might be a product or service name (e.g., ExampleCache or "Example CDN"), a hostname ("cache-3.example.com"), an IP address, or a generated string.

Each member of the list can have parameters that describe that cache's handling of the request. While these parameters are OPTIONAL, caches are encouraged to provide as much information as possible.

This specification defines the following parameters:

hit	= sf-boolean
fwd	= sf-token
fwd-status	= sf-integer
ttl	= sf-integer
stored	= sf-boolean
collapsed	= sf-boolean
key	= sf-string
detail	= sf-token / sf-string

### 2.1. The hit parameter

"hit", when true, indicates that the request was satisfied by the cache; i.e., it was not forwarded, and the response was obtained from the cache.

A response that was originally produced by the origin but was modified by the cache (for example, a 304 or 206 status code) is still considered a hit, as long as it did not go forward (e.g., for validation).

A response that was in cache but not able to be used without going forward (e.g., because it was stale, or partial) is not considered a hit. Note that a stale response that is used without going forward (e.g., because the origin server is not available) can be considered a hit.

"hit" and "fwd" are exclusive; only one of them should appear on each list member.

### 2.2. The fwd parameter

"fwd" indicates that the request went forward towards the origin, and why.

The following parameter values are defined to explain why the request went forward, from most specific to least:

- \* bypass - The cache was configured to not handle this request
- \* method - The request method's semantics require the request to be forwarded
- \* uri-miss - The cache did not contain any responses that matched the request URI
- \* vary-miss - The cache contained a response that matched the request URI, but could not select a response based upon this request's headers and stored Vary headers.
- \* miss - The cache did not contain any responses that could be used to satisfy this request (to be used when an implementation cannot distinguish between uri-miss and vary-miss)
- \* request - The cache was able to select a fresh response for the request, but the request's semantics (e.g., Cache-Control request directives) did not allow its use
- \* stale - The cache was able to select a response for the request, but it was stale
- \* partial - The cache was able to select a partial response for the request, but it did not contain all of the requested ranges (or the request was for the complete response)

The most specific reason that the cache is aware of SHOULD be used, to the extent that it is possible to implement. See also [HTTP-CACHING], Section 4.

### 2.3. The fwd-status parameter

"fwd-status" indicates what status code (see [HTTP], Section 15) the next hop server returned in response to the forwarded request. Only meaningful when "fwd" is present; if "fwd-status" is not present but "fwd" is, it defaults to the status code sent in the response.

This parameter is useful to distinguish cases when the next hop server sends a 304 Not Modified response to a conditional request, or a 206 Partial Response because of a range request.

#### 2.4. The ttl parameter

"ttl" indicates the response's remaining freshness lifetime (see [HTTP-CACHING], Section 4.2.1) as calculated by the cache, as an integer number of seconds, measured as closely as possible to when the response header section is sent by the cache. This includes freshness assigned by the cache; e.g., through heuristics (see [HTTP-CACHING], Section 4.2.2), local configuration, or other factors. May be negative, to indicate staleness.

#### 2.5. The stored parameter

"stored" indicates whether the cache stored the response (see [HTTP-CACHING], Section 3); a true value indicates that it did. Only meaningful when fwd is present.

#### 2.6. The collapsed parameter

"collapsed" indicates whether this request was collapsed together with one or more other forward requests (see [HTTP-CACHING], Section 4); if true, the response was successfully reused; if not, a new request had to be made. If not present, the request was not collapsed with others. Only meaningful when fwd is present.

#### 2.7. The key parameter

"key" conveys a representation of the cache key (see [HTTP-CACHING], Section 2) used for the response. Note that this may be implementation-specific.

#### 2.8. The detail parameter

"detail" allows implementations to convey additional information not captured in other parameters; for example, implementation-specific states, or other caching-related metrics.

For example:

```
Cache-Status: ExampleCache; hit; detail=MEMORY
```

The semantics of a detail parameter are always specific to the cache that sent it; even if a member of details from another cache shares the same name, it might not mean the same thing.

This parameter is intentionally limited. If an implementation's developer or operator needs to convey additional information in an interoperable fashion, they are encouraged to register extension parameters (see Section 4) or define another header field.

### 3. Examples

The most minimal cache hit:

```
Cache-Status: ExampleCache; hit
```

... but a polite cache will give some more information, e.g.:

```
Cache-Status: ExampleCache; hit; ttl=376
```

A stale hit just has negative freshness:

```
Cache-Status: ExampleCache; hit; ttl=-412
```

Whereas a complete miss is:

```
Cache-Status: ExampleCache; fwd=uri-miss
```

A miss that successfully validated on the back-end server:

```
Cache-Status: ExampleCache; fwd=stale; fwd-status=304
```

A miss that was collapsed with another request:

```
Cache-Status: ExampleCache; fwd=uri-miss; collapsed
```

A miss that the cache attempted to collapse, but couldn't:

```
Cache-Status: ExampleCache; fwd=uri-miss; collapsed=?0
```

Going through two separate layers of caching, where the cache closest to the origin responded to an earlier request with a stored response, and a second cache stored that response and later reused it to satisfy the current request:

```
Cache-Status: OriginCache; hit; ttl=1100,  
             "CDN Company Here"; hit; ttl=545
```

Going through a three-layer caching system, where the closest to the origin is a reverse proxy (where the response was served from cache), the next is a forward proxy interposed by the network (where the request was forwarded because there wasn't any response cached with its URI, the request was collapsed with others, and the resulting response was stored), and the closest to the user is a browser cache (where there wasn't any response cached with the request's URI):

```
Cache-Status: ReverseProxyCache; hit
Cache-Status: ForwardProxyCache; fwd=uri-miss; collapsed; stored
Cache-Status: BrowserCache; fwd=uri-miss
```

#### 4. Defining New Cache-Status Parameters

New Cache-Status Parameters can be defined by registering them in the HTTP Cache-Status Parameters registry.

Registration requests are reviewed and approved by a Designated Expert, as per [RFC8126], Section 4.5. A specification document is appreciated, but not required.

The Expert(s) should consider the following factors when evaluating requests:

- \* Community feedback
- \* If the value is sufficiently well-defined
- \* Generic parameters are preferred over vendor-specific, application-specific, or deployment-specific values. If a generic value cannot be agreed upon in the community, the parameter's name should be correspondingly specific (e.g., with a prefix that identifies the vendor, application or deployment).

Registration requests should use the following template:

- \* Name: [a name for the Cache-Status Parameter that matches the 'key' ABNF rule]
- \* Description: [a description of the parameter semantics and value]
- \* Reference: [to a specification defining this parameter, if available]

See the registry at <https://iana.org/assignments/http-cache-status> (<https://iana.org/assignments/http-cache-status>) for details on where to send registration requests.

#### 5. IANA Considerations

Upon publication, please create the HTTP Cache-Status Parameters registry at <https://iana.org/assignments/http-cache-status> (<https://iana.org/assignments/http-cache-status>) and populate it with the types defined in Section 2; see Section 4 for its associated procedures.

Also, please create the following entry in the Hypertext Transfer Protocol (HTTP) Field Name Registry defined in [HTTP], Section 18.4:

- \* Field name: Cache-Status
- \* Status: permanent
- \* Specification document: [this document]
- \* Comments:

## 6. Security Considerations

Attackers can use the information in Cache-Status to probe the behaviour of the cache (and other components), and infer the activity of those using the cache. The Cache-Status header field may not create these risks on its own, but can assist attackers in exploiting them.

For example, knowing if a cache has stored a response can help an attacker execute a timing attack on sensitive data.

Additionally, exposing the cache key can help an attacker understand modifications to the cache key, which may assist cache poisoning attacks. See [ENTANGLE] for details.

The underlying risks can be mitigated with a variety of techniques (e.g., use of encryption and authentication; avoiding the inclusion of attacker-controlled data in the cache key), depending on their exact nature. Note that merely obfuscating the key does not mitigate this risk.

To avoid assisting such attacks, the Cache-Status header field can be omitted, only sent when the client is authorized to receive it, or only send sensitive information (e.g., the key parameter) when the client is authorized.

## 7. References

### 7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.



- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.
- [STRUCTURED-FIELDS] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/rfc/rfc8941>>.
- [HTTP] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-17, 25 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-17>>.
- [HTTP-CACHING] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Caching", Work in Progress, Internet-Draft, draft-ietf-httpbis-cache-17, 25 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cache-17>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/rfc/rfc5234>>.

## 7.2. Informative References

- [ENTANGLE] Kettle, J., "Web Cache Entanglement: Novel Pathways to Poisoning", 2020, <<https://i.blackhat.com/USA-20/Wednesday/us-20-Kettle-Web-Cache-Entanglement-Novel-Pathways-To-Poisoning-wp.pdf>>.

### Author's Address

Mark Nottingham  
Fastly  
Pahran VIC  
Australia

Email: [mnot@mnot.net](mailto:mnot@mnot.net)  
URI: <https://www.mnot.net/>

HTTP Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: January 4, 2021

I. Grigorik  
Y. Weiss  
Google  
July 3, 2020

HTTP Client Hints  
draft-ietf-httpbis-client-hints-15

Abstract

HTTP defines proactive content negotiation to allow servers to select the appropriate response for a given request, based upon the user agent's characteristics, as expressed in request headers. In practice, user agents are often unwilling to send those request headers, because it is not clear whether they will be used, and sending them impacts both performance and privacy.

This document defines an Accept-CH response header that servers can use to advertise their use of request headers for proactive content negotiation, along with a set of guidelines for the creation of such headers, colloquially known as "Client Hints."

Note to Readers

Discussion of this draft takes place on the HTTP working group mailing list ([ietf-http-wg@w3.org](mailto:ietf-http-wg@w3.org)), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/> [1].

Working Group information can be found at <http://httpwg.github.io/> [2]; source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions/labels/client-hints> [3].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 4, 2021.

## Copyright Notice

Copyright (c) 2020 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 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 . . . . .	3
1.1. Notational Conventions . . . . .	4
2. Client Hint Request Header Fields . . . . .	4
2.1. Sending Client Hints . . . . .	4
2.2. Server Processing of Client Hints . . . . .	5
3. Advertising Server Support . . . . .	5
3.1. The Accept-CH Response Header Field . . . . .	5
3.2. Interaction with Caches . . . . .	6
4. Security Considerations . . . . .	7
4.1. Information Exposure . . . . .	7
4.2. Deployment and Security Risks . . . . .	9
4.3. Abuse Detection . . . . .	9
5. Cost of Sending Hints . . . . .	9
6. IANA Considerations . . . . .	10
6.1. Accept-CH . . . . .	10
7. References . . . . .	10
7.1. Normative References . . . . .	10
7.2. Informative References . . . . .	11
7.3. URIs . . . . .	11
Appendix A. Changes . . . . .	11
A.1. Since -00 . . . . .	11
A.2. Since -01 . . . . .	12
A.3. Since -02 . . . . .	12
A.4. Since -03 . . . . .	12
A.5. Since -04 . . . . .	12
A.6. Since -05 . . . . .	12
A.7. Since -06 . . . . .	12
A.8. Since -07 . . . . .	12
A.9. Since -08 . . . . .	13

A.10. Since -09 . . . . .	13
A.11. Since -10 . . . . .	13
A.12. Since -11 . . . . .	13
A.13. Since -12 . . . . .	13
A.14. Since -13 . . . . .	13
A.15. Since -14 . . . . .	13
Acknowledgements . . . . .	13
Authors' Addresses . . . . .	13

## 1. Introduction

There are thousands of different devices accessing the web, each with different device capabilities and preference information. These device capabilities include hardware and software characteristics, as well as dynamic user and user agent preferences. Historically, applications that wanted the server to optimize content delivery and user experience based on such capabilities had to rely on passive identification (e.g., by matching the User-Agent header field (Section 5.5.3 of [RFC7231]) against an established database of user agent signatures), use HTTP cookies [RFC6265] and URL parameters, or use some combination of these and similar mechanisms to enable ad hoc content negotiation.

Such techniques are expensive to set up and maintain, and are not portable across both applications and servers. They also make it hard for both user agent and server to understand which data are required and is in use during the negotiation:

- o User agent detection cannot reliably identify all static variables, cannot infer dynamic user agent preferences, requires an external device database, is not cache friendly, and is reliant on a passive fingerprinting surface.
- o Cookie-based approaches are not portable across applications and servers, impose additional client-side latency by requiring JavaScript execution, and are not cache friendly.
- o URL parameters, similar to cookie-based approaches, suffer from lack of portability, and are hard to deploy due to a requirement to encode content negotiation data inside of the URL of each resource.

Proactive content negotiation (Section 3.4.1 of [RFC7231]) offers an alternative approach; user agents use specified, well-defined request headers to advertise their capabilities and characteristics, so that servers can select (or formulate) an appropriate response based on those request headers (or on other, implicit characteristics).

However, traditional proactive content negotiation techniques often mean that user agents send these request headers prolifically. This

causes performance concerns (because it creates "bloat" in requests), as well as privacy issues; passively providing such information allows servers to silently fingerprint the user.

This document defines Client Hints, a framework that enables servers to opt-in to specific proactive content negotiation features, adapting their content accordingly, as well as guidelines for content negotiation mechanisms that use the framework. This document also defines a new response header, Accept-CH, that allows an origin server to explicitly ask that user agents send these headers in requests.

Client Hints mitigate performance concerns by assuring that user agents will only send the request headers when they're actually going to be used, and privacy concerns of passive fingerprinting by requiring explicit opt-in and disclosure of required headers by the server through the use of the Accept-CH response header, turning passive fingerprinting vectors into active ones.

The document does not define specific usages of Client Hints. Such usages need to be defined in their respective specifications.

One example of such usage is the User Agent Client Hints [UA-CH].

### 1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

## 2. Client Hint Request Header Fields

A Client Hint request header field is a HTTP header field that is used by HTTP user agents to indicate data that can be used by the server to select an appropriate response. Each one conveys user agent preferences that the server can use to adapt and optimize the response.

### 2.1. Sending Client Hints

User agents choose what Client Hints to send in a request based on their default settings, user configuration, and server preferences expressed in "Accept-CH". The user agent and server can use an opt-

in mechanism outlined below to negotiate which header fields need to be sent to allow for efficient content adaption, and optionally use additional mechanisms (e.g., as outlined in [CLIENT-HINTS-INFRASTRUCTURE]) to negotiate delegation policies that control access of third parties to those same header fields. User agents SHOULD require an opt-in to send any hints that are not listed in the low-entropy hint table at [CLIENT-HINTS-INFRASTRUCTURE].

Implementers need to be aware of the fingerprinting implications when implementing support for Client Hints, and follow the considerations outlined in the Security Considerations (Section 4) section of this document.

## 2.2. Server Processing of Client Hints

When presented with a request that contains one or more Client Hint header fields, servers can optimize the response based upon the information in them. When doing so, and if the resource is cacheable, the server MUST also generate a Vary response header field (Section 7.1.4 of [RFC7231]) to indicate which hints can affect the selected response and whether the selected response is appropriate for a later request.

Servers MUST ignore hints they do not understand nor support. There is no mechanism for servers to indicate to user agents that hints were ignored.

Furthermore, the server can generate additional response header fields (as specified by the hint or hints in use) that convey related values to aid client processing.

## 3. Advertising Server Support

Servers can advertise support for Client Hints using the mechanism described below.

### 3.1. The Accept-CH Response Header Field

The Accept-CH response header field indicates server support for the hints indicated in its value. Servers wishing to receive user agent information through Client Hints SHOULD add Accept-CH response header to their responses as early as possible.

Accept-CH is a Structured Header [I-D.ietf-httpbis-header-structure]. Its value MUST be an sf-list (Section 3.1 of [I-D.ietf-httpbis-header-structure]) whose members are tokens (Section 3.3.4 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

Accept-CH = sf-list

For example:

Accept-CH: Sec-CH-Example, Sec-CH-Example-2

When a user agent receives an HTTP response containing "Accept-CH", that indicates that the origin opts-in to receive the indicated request header fields for subsequent same-origin requests. The opt-in MUST be ignored if delivered over non-secure transport (using a scheme different from HTTPS). It SHOULD be persisted and bound to the origin to enable delivery of Client Hints on subsequent requests to the server's origin, for the duration of the user's session (as defined by the user agent). An opt-in overrides previous persisted opt-in values and SHOULD be persisted in its stead.

Based on the Accept-CH example above, which is received in response to a user agent navigating to "https://site.example", and delivered over a secure transport, persisted Accept-CH preferences will be bound to "https://site.example". It will then use it for navigations to e.g., "https://site.example/foobar.html", but not to e.g., "https://foobar.site.example/". It will similarly use the preference for any same-origin resource requests (e.g., to "https://site.example/image.jpg") initiated by the page constructed from the navigation's response, but not to cross-origin resource requests (e.g., "https://thirdparty.example/resource.js"). This preference will not extend to resource requests initiated to "https://site.example" from other origins (e.g., from navigations to "https://other.example/").

### 3.2. Interaction with Caches

When selecting a response based on one or more Client Hints, and if the resource is cacheable, the server needs to generate a Vary response header field ([RFC7234]) to indicate which hints can affect the selected response and whether the selected response is appropriate for a later request.

Vary: Sec-CH-Example

The above example indicates that the cache key needs to include the Sec-CH-Example header field.

Vary: Sec-CH-Example, Sec-CH-Example-2

The above example indicates that the cache key needs to include the Sec-CH-Example and Sec-CH-Example-2 header fields.

## 4. Security Considerations

### 4.1. Information Exposure

Request header fields used in features relying on this document expose information about the user's environment to enable privacy-preserving proactive content negotiation, and avoid exposing passive fingerprinting vectors. However, implementers need to bear in mind that in the worst case, uncontrolled and unmonitored active fingerprinting is not better than passive fingerprinting. In order to provide user privacy benefits, user agents need to apply further policies that prevent abuse of the information exposed by features using Client Hints.

The information exposed by features might reveal new information about the user and implementers ought to consider the following considerations, recommendations, and best practices.

The underlying assumption is that exposing information about the user as a request header is equivalent (from a security perspective) to exposing this information by other means. (For example, if the request's origin can access that information using JavaScript APIs, and transmit it to its servers).

Because Client Hints is an explicit opt-in mechanism, that means that servers that want access to information about the user's environment need to actively ask for it, enabling clients and privacy researchers to keep track of which origins collect that data, and potentially act upon it. The header-based opt-in means that removal of passive fingerprinting vectors is possible, such as the User-Agent string (enabling active access to that information through User-Agent Client Hints ([UA-CH]) or otherwise expose information already available through script (e.g., the Save-Data Client Hint [4]), without increasing the passive fingerprinting surface. User agents supporting Client Hints features which send certain information to opted-in servers SHOULD avoid sending the equivalent information passively.

Therefore, features relying on this document to define Client Hint headers MUST NOT provide new information that is otherwise not made available to the application by the user agent, such as existing request headers, HTML, CSS, or JavaScript.

Such features need to take into account the following aspects of the information exposed:

- o Entropy - Exposing highly granular data can be used to help identify users across multiple requests to different origins.



Reducing the set of header field values that can be expressed, or restricting them to an enumerated range where the advertised value is close to but is not an exact representation of the current value, can improve privacy and reduce risk of linkability by ensuring that the same value is sent by multiple users.

- o Sensitivity - The feature SHOULD NOT expose user-sensitive information. To that end, information available to the application, but gated behind specific user actions (e.g., a permission prompt or user activation) SHOULD NOT be exposed as a Client Hint.
- o Change over time - The feature SHOULD NOT expose user information that changes over time, unless the state change itself is also exposed (e.g., through JavaScript callbacks).

Different features will be positioned in different points in the space between low-entropy, non-sensitive and static information (e.g., user agent information), and high-entropy, sensitive and dynamic information (e.g., geolocation). User agents need to consider the value provided by a particular feature vs these considerations, and may wish to have different policies regarding that tradeoff on a per-feature or other fine-grained basis.

Implementers ought to consider both user- and server- controlled mechanisms and policies to control which Client Hints header fields are advertised:

- o Implementers SHOULD restrict delivery of some or all Client Hints header fields to the opt-in origin only, unless the opt-in origin has explicitly delegated permission to another origin to request Client Hints header fields.
- o Implementers considering providing user choice mechanisms that allow users to balance privacy concerns against bandwidth limitations need to also consider that explaining to users the privacy implications involved, such as the risks of passive fingerprinting, may be challenging or even impractical.
- o Implementations specific to certain use cases or threat models MAY avoid transmitting some or all of Client Hints header fields. For example, avoid transmission of header fields that can carry higher risks of linkability.

User agents MUST clear persisted opt-in preferences when any one of site data, browsing history, browsing cache, cookies, or similar, are cleared.

#### 4.2. Deployment and Security Risks

Deployment of new request headers requires several considerations:

- o Potential conflicts due to existing use of header field name
- o Properties of the data communicated in header field value

Authors of new Client Hints are advised to carefully consider whether they need to be able to be added by client-side content (e.g., scripts), or whether they need to be exclusively set by the user agent. In the latter case, the Sec- prefix on the header field name has the effect of preventing scripts and other application content from setting them in user agents. Using the "Sec-" prefix signals to servers that the user agent - and not application content - generated the values. See [FETCH] for more information.

By convention, request headers that are Client Hints are encouraged to use a CH- prefix, to make them easier to identify as using this framework; for example, CH-Foo or, with a "Sec-" prefix, Sec-CH-Foo. Doing so makes them easier to identify programmatically (e.g., for stripping unrecognised hints from requests by privacy filters).

A Client Hints request header negotiated using the Accept-CH opt-in mechanism MUST have a field name that matches sf-token (Section 3.3.4 of [I-D.ietf-httpbis-header-structure]).

#### 4.3. Abuse Detection

A user agent that tracks access to active fingerprinting information SHOULD consider emission of Client Hints headers similarly to the way it would consider access to the equivalent API.

Research into abuse of Client Hints might look at how HTTP responses to requests that contain Client Hints differ from those with different values, and from those without. This might be used to reveal which Client Hints are in use, allowing researchers to further analyze that use.

#### 5. Cost of Sending Hints

Sending Client Hints to the server incurs an increase in request byte size. Some of this increase can be mitigated by HTTP header compression schemes, but each new hint sent will still lead to some increased bandwidth usage. Servers SHOULD take that into account when opting in to receive Client Hints, and SHOULD NOT opt-in to receive hints unless they are to be used for content adaptation purposes.

Due to request byte size increase, features relying on this document to define Client Hints MAY consider restricting sending those hints to certain request destinations [FETCH], where they are more likely to be useful.

## 6. IANA Considerations

Features relying on this document are expected to register added request header fields in the Permanent Message Header Fields registry ([RFC3864]).

This document defines the "Accept-CH" HTTP response header field, and registers it in the same registry.

### 6.1. Accept-CH

- o Header field name: Accept-CH
- o Applicable protocol: HTTP
- o Status: experimental
- o Author/Change controller: IETF
- o Specification document(s): Section 3.1 of this document
- o Related information: for Client Hints

## 7. References

### 7.1. Normative References

- [CLIENT-HINTS-INFRASTRUCTURE]  
Weiss, Y., "Client Hints Infrastructure", n.d.,  
<<https://wicg.github.io/client-hints-infrastructure/>>.
- [I-D.ietf-httpbis-header-structure]  
Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", draft-ietf-httpbis-header-structure-19 (work in progress), June 2020.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <<https://www.rfc-editor.org/info/rfc3864>>.

- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/info/rfc7234>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 7.2. Informative References

- [FETCH] van Kesteren, A., "Fetch", n.d., <<https://fetch.spec.whatwg.org/>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [UA-CH] West, M. and Y. Weiss, "User Agent Client Hints", n.d., <<https://wicg.github.io/ua-client-hints/>>.

## 7.3. URIs

- [1] <https://lists.w3.org/Archives/Public/ietf-http-wg/>
- [2] <http://httpwg.github.io/>
- [3] <https://github.com/httpwg/http-extensions/labels/client-hints>
- [4] <https://wicg.github.io/savedata/#save-data-request-header-field>

## Appendix A. Changes

### A.1. Since -00

- o Issue 168 (make Save-Data extensible) updated ABNF.
- o Issue 163 (CH review feedback) editorial feedback from httpwg list.

- o Issue 153 (NetInfo API citation) added normative reference.
- A.2. Since -01
- o Issue 200: Moved Key reference to informative.
  - o Issue 215: Extended passive fingerprinting and mitigation considerations.
  - o Changed document status to experimental.
- A.3. Since -02
- o Issue 239: Updated reference to CR-css-values-3
  - o Issue 240: Updated reference for Network Information API
  - o Issue 241: Consistency in IANA considerations
  - o Issue 250: Clarified Accept-CH
- A.4. Since -03
- o Issue 284: Extended guidance for Accept-CH
  - o Issue 308: Editorial cleanup
  - o Issue 306: Define Accept-CH-Lifetime
- A.5. Since -04
- o Issue 361: Removed Downlink
  - o Issue 361: Moved Key to appendix, plus other editorial feedback
- A.6. Since -05
- o Issue 372: Scoped CH opt-in and delivery to secure transports
  - o Issue 373: Bind CH opt-in to origin
- A.7. Since -06
- o Issue 524: Save-Data is now defined by NetInfo spec, dropping
  - o PR 775: Removed specific features to be defined in other specifications
- A.8. Since -07
- o Issue 761: Clarified that the defined headers are response headers.
  - o Issue 730: Replaced Key reference with Variants.
  - o Issue 700: Replaced ABNF with structured headers.
  - o PR 878: Removed Accept-CH-Lifetime based on feedback at IETF 105

## A.9. Since -08

- o PR 985: Describe the bytesize cost of hints.
- o PR 776: Add Sec- and CH- prefix considerations.
- o PR 1001: Clear CH persistence when cookies are cleared.

## A.10. Since -09

- o PR 1064: Fix merge issues with "cost of sending hints".

## A.11. Since -10

- o PR 1072: LC feedback from Julian Reschke.
- o PR 1080: Improve list style.
- o PR 1082: Remove section mentioning Variants.
- o PR 1097: Editorial feedback from mnot.
- o PR 1131: Remove unused references.
- o PR 1132: Remove nested list.

## A.12. Since -11

- o PR 1134: Re-insert back section.

## A.13. Since -12

- o PR 1160: AD review.

## A.14. Since -13

- o PR 1171: Genart review.

## A.15. Since -14

- o PR 1220: AD review.

## Acknowledgements

Thanks to Mark Nottingham, Julian Reschke, Chris Bentzel, Ben Greenstein, Tarun Bansal, Roy Fielding, Vasiliy Faronov, Ted Hardie, Jonas Sicking, Martin Thomson, and numerous other members of the IETF HTTP Working Group for invaluable help and feedback.

## Authors' Addresses

Ilya Grigorik  
Google

Email: [ilya@igvita.com](mailto:ilya@igvita.com)  
URI: <https://www.igvita.com/>

Yoav Weiss  
Google

Email: [yoav@yoav.ws](mailto:yoav@yoav.ws)  
URI: <https://blog.yoav.ws/>

HTTP	R. Polli
Internet-Draft	Team Digitale, Italian Government
Obsoletes: 3230 (if approved)	L. Pardue
Intended status: Standards Track	Cloudflare
Expires: 22 September 2022	21 March 2022

Digest Fields  
draft-ietf-httpbis-digest-headers-08

## Abstract

This document defines HTTP fields that support integrity digests. The Repr-Digest field can be used for the integrity of HTTP representations. The Content-Digest field can be used for the integrity of HTTP message content. Want-Repr-Digest and Want-Content-Digest can be used to indicate a sender's interest and preferences for receiving the respective Integrity fields.

This document obsoletes RFC 3230 and the Digest and Want-Digest HTTP fields.

## About This Document

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

Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-httpbis-digest-headers/>.

Discussion of this document takes place on the HTTP Working Group mailing list (<mailto:ietf-http-wg@w3.org>), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/>. Working Group information can be found at <https://httpwg.org/>.

Source for this draft and an issue tracker can be found at <https://github.com/httpwg/http-extensions/labels/digest-headers>.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.



Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 22 September 2022.

## 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 . . . . .	3
1.1. Document Structure . . . . .	4
1.2. Concept Overview . . . . .	4
1.3. Obsoleting RFC 3230 . . . . .	5
1.4. Notational Conventions . . . . .	6
2. The Repr-Digest Field . . . . .	7
2.1. Using Repr-Digest in State-Changing Requests . . . . .	8
2.2. Repr-Digest and Content-Location in Responses . . . . .	9
3. The Content-Digest Field . . . . .	9
4. Integrity preference fields . . . . .	10
5. Hash Algorithms for HTTP Digest Fields Registry . . . . .	11
6. Security Considerations . . . . .	13
6.1. HTTP Messages Are Not Protected In Full . . . . .	13
6.2. End-to-End Integrity . . . . .	13
6.3. Usage in Signatures . . . . .	13
6.4. Usage in Trailer Fields . . . . .	14
6.5. Usage with Encryption . . . . .	14
6.6. Algorithm Agility . . . . .	14
6.7. Resource exhaustion . . . . .	15
7. IANA Considerations . . . . .	15
7.1. HTTP Field Name Registration . . . . .	15
7.2. Establish the Hash Algorithms for HTTP Digest Fields Registry . . . . .	16
8. References . . . . .	16
8.1. Normative References . . . . .	16

8.2. Informative References . . . . .	18
Appendix A. Resource Representation and Representation Data . . .	19
Appendix B. Examples of Unsolicited Digest . . . . .	21
B.1. Server Returns Full Representation Data . . . . .	21
B.2. Server Returns No Representation Data . . . . .	22
B.3. Server Returns Partial Representation Data . . . . .	22
B.4. Client and Server Provide Full Representation Data . . .	23
B.5. Client Provides Full Representation Data, Server Provides No Representation Data . . . . .	24
B.6. Client and Server Provide Full Representation Data . . .	25
B.7. POST Response does not Reference the Request URI . . . .	25
B.8. POST Response Describes the Request Status . . . . .	26
B.9. Digest with PATCH . . . . .	27
B.10. Error responses . . . . .	28
B.11. Use with Trailer Fields and Transfer Coding . . . . .	28
Appendix C. Examples of Want-Repr-Digest Solicited Digest . . .	29
C.1. Server Selects Client's Least Preferred Algorithm . . . .	29
C.2. Server Selects Algorithm Unsupported by Client . . . . .	30
C.3. Server Does Not Support Client Algorithm and Returns an Error . . . . .	30
Acknowledgements . . . . .	31
Code Samples . . . . .	31
Changes . . . . .	32
Since draft-ietf-httpbis-digest-headers-06 . . . . .	32
Since draft-ietf-httpbis-digest-headers-05 . . . . .	33
Since draft-ietf-httpbis-digest-headers-04 . . . . .	33
Since draft-ietf-httpbis-digest-headers-03 . . . . .	33
Since draft-ietf-httpbis-digest-headers-02 . . . . .	33
Since draft-ietf-httpbis-digest-headers-01 . . . . .	34
Since draft-ietf-httpbis-digest-headers-00 . . . . .	34
Authors' Addresses . . . . .	34

## 1. Introduction

HTTP does not define the means to protect the data integrity of representations or content. When HTTP messages are transferred between endpoints, lower layer features or properties such as TCP checksums or TLS records [RFC2818] can provide some integrity protection. However, transport-oriented integrity provides a limited utility because it is opaque to the application layer and only covers the extent of a single connection. HTTP messages often travel over a chain of separate connections, in between connections there is a possibility for unintended or malicious data corruption. An HTTP integrity mechanism can provide the means for endpoints, or applications using HTTP, to detect data corruption and make a choice about how to act on it. An example use case is to aid fault detection and diagnosis across system boundaries.

This document defines two digest integrity mechanisms for HTTP. First, representation data integrity, which acts on representation data (Section 3.2 of [SEMANTICS]). This supports advanced use cases such as validating the integrity of a resource that was reconstructed from parts retrieved using multiple requests or connections. Second, content integrity, which acts on conveyed content (Section 6.4 of [SEMANTICS]).

This document obsoletes RFC 3230 and therefore the Digest and Want-Digest HTTP fields; see Section 1.3.

### 1.1. Document Structure

This document is structured as follows:

- \* Section 2 defines the Repr-Digest request and response header and trailer field,
- \* Section 3 defines the Content-Digest request and response header and trailer field,
- \* Section 4 defines the Want-Repr-Digest and Want-Content-Digest request and response header and trailer field,
- \* Section 5 describes algorithms and their relation to the fields defined in this document,
- \* Section 2.1 details computing representation digests,
- \* Appendix B and Appendix C provide examples of using Repr-Digest and Want-Repr-Digest.

### 1.2. Concept Overview

The HTTP fields defined in this document can be used for HTTP integrity. Senders choose a hashing algorithm and calculate a digest from an input related to the HTTP message, the algorithm identifier and digest are transmitted in an HTTP field. Receivers can validate the digest for integrity purposes. Hashing algorithms are registered in the "Hash Algorithms for HTTP Digest Fields" (see Section 5).

Selecting the data on which digests are calculated depends on the use case of HTTP messages. This document provides different headers for HTTP representation data and HTTP content.

This document defines the Repr-Digest request and response header and trailer field (Section 2) that contains a digest value computed by applying a hashing algorithm to "selected representation data"

(Section 3.2 of [SEMANTICS]). Basing Repr-Digest on the selected representation makes it straightforward to apply it to use-cases where the transferred data requires some sort of manipulation to be considered a representation or conveys a partial representation of a resource, such as Range Requests (see Section 14.2 of [SEMANTICS]).

There are use-cases where a simple digest of the HTTP content bytes is required. The Content-Digest request and response header and trailer field is defined to support digests of content (Section 3.2 of [SEMANTICS]); see Section 3.

Repr-Digest and Content-Digest support hashing algorithm agility. The Want-Repr-Digest and Want-Content-Digest fields allows endpoints to express interest in Repr-Digest and Content-Digest respectively, and preference of algorithms in either.

Repr-Digest and Content-Digest are collectively termed Integrity fields. Want-Repr-Digest and Want-Content-Digest are collectively termed Integrity preference fields.

Integrity fields are tied to the Content-Encoding and Content-Type header fields. Therefore, a given resource may have multiple different digest values when transferred with HTTP.

Integrity fields do not provide integrity for HTTP messages or fields. However, they can be combined with other mechanisms that protect metadata, such as digital signatures, in order to protect the phases of an HTTP exchange in whole or in part.

This specification does not define means for authentication, authorization or privacy.

### 1.3. Obsoleting RFC 3230

[RFC3230] defined the Digest and Want-Digest HTTP fields for HTTP integrity. It also coined the term "instance" and "instance manipulation" in order to explain concepts that are now more universally defined, and implemented, as HTTP semantics such as "selected representation data" (Section 3.2 of [SEMANTICS]).

Experience has shown that implementations of [RFC3230] have interpreted the meaning of "instance" inconsistently, leading to interoperability issues. The most common mistake being the calculation of the digest using (what we now call) message content, rather than using (what we now call) representation data as was originally intended. Interestingly, time has also shown that a digest of message content can be beneficial for some use cases. So it is difficult to detect if non-conformance to [RFC3230] is intentional or unintentional.

In order to address potential inconsistencies and ambiguity across implementations of Digest and Want-Digest, this document obsoletes [RFC3230]. The Integrity fields (Section 2 and Section 3) and Integrity preference fields (Section 4) defined in this document are better aligned with current HTTP semantics and have names that more clearly articulate the intended usages.

#### 1.4. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the Augmented BNF defined in [RFC5234] and updated by [RFC7405].

This document uses the Boolean, Byte Sequence, Dictionary, Integer and List types from [STRUCTURED-FIELDS] along with the sf-dictionary and sf-list ABNF rules.

The definitions "representation", "selected representation", "representation data", "representation metadata", "user agent" and "content" in this document are to be interpreted as described in [SEMANTICS].

Hashing algorithm names respect the casing used in their definition document (e.g. SHA-1, CRC32c) whereas hashing algorithm keys are quoted (e.g. "sha", "crc32c").

The term "checksum" describes the output of the application of an algorithm to a sequence of bytes, whereas "digest" is only used in relation to the value contained in the fields.

Integrity fields: collective term for Repr-Digest and Content-Digest

Integrity preference fields: collective term for Want-Repr-Digest and Want-Content-Digest

## 2. The Repr-Digest Field

The Repr-Digest HTTP field can be used in requests and responses to communicate digests that are calculated using a hashing algorithm applied to the entire "selected representation data" (see Section 8.1 of [SEMANTICS]).

Representations take into account the effect of the HTTP semantics on messages. For example, the content can be affected by Range Requests or methods such as HEAD, while the way the content is transferred "on the wire" is dependent on other transformations (e.g. transfer codings for HTTP/1.1 - see Section 6.1 of [HTTP11]). To help illustrate HTTP representation concepts, several examples are provided in Appendix A.

When a message has no "representation data" it is still possible to assert that no "representation data" was sent by computing the digest on an empty string (see Section 6.3).

Repr-Digest is a Structured Fields Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]) where:

- \* keys convey the hashing algorithm (see Section 5) used to compute the digest;
- \* values MUST be of type Byte Sequence, which contain the output of the digest calculation.

Repr-Digest = sf-dictionary

For example:

NOTE: '\ ' line wrapping per RFC 8792

```
Repr-Digest: \
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==;
```

The Dictionary type can be used, for example, to attach multiple digests calculated using different hashing algorithms in order to support a population of endpoints with different or evolving capabilities. Such an approach could support transitions away from weaker algorithms (see Section 6.6).

NOTE: '\' line wrapping per RFC 8792

```
Repr-Digest: \
  sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=:\
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:
```

A recipient MAY ignore any or all digests. This allows the recipient to choose which hashing algorithm(s) to use for validation instead of verifying every digest.

A sender MAY send a digest without knowing whether the recipient supports a given hashing algorithm, or even knowing that the recipient will ignore it.

Repr-Digest can be sent in a trailer section. In this case, Repr-Digest MAY be merged into the header section; see Section 6.5.1 of [SEMANTICS].

## 2.1. Using Repr-Digest in State-Changing Requests

When the representation enclosed in a state-changing request does not describe the target resource, the representation digest MUST be computed on the representation data. This is the only possible choice because representation digest requires complete representation metadata (see Section 2).

In responses,

- \* if the representation describes the status of the request, Repr-Digest MUST be computed on the enclosed representation (see Appendix B.8 );
- \* if there is a referenced resource Repr-Digest MUST be computed on the selected representation of the referenced resource even if that is different from the target resource. That might or might not result in computing Repr-Digest on the enclosed representation.

The latter case is done according to the HTTP semantics of the given method, for example using the Content-Location header field (see Section 8.7 of [SEMANTICS]). In contrast, the Location header field does not affect Repr-Digest because it is not representation metadata.

For example, in PATCH requests, the representation digest will be computed on the patch document because the representation metadata refers to the patch document and not to the target resource (see

Section 2 of [PATCH]). In responses, instead, the representation digest will be computed on the selected representation of the patched resource.

## 2.2. Repr-Digest and Content-Location in Responses

When a state-changing method returns the Content-Location header field, the enclosed representation refers to the resource identified by its value and Repr-Digest is computed accordingly. An example is given in Appendix B.7.

## 3. The Content-Digest Field

The Content-Digest HTTP field can be used in requests and responses to communicate digests that are calculated using a hashing algorithm applied to the actual message content (see Section 6.4 of [SEMANTICS]). It is a Structured Fields Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]) where:

- \* keys convey the hashing algorithm (see Section 5) used to compute the digest;
- \* values MUST be Byte Sequences (Section 3.3.5 of [STRUCTURED-FIELDS]) containing the output of the digest calculation.

Content-Digest = sf-dictionary

For example:

NOTE: '\ ' line wrapping per RFC 8792

```
Content-Digest: \
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:
```

The Dictionary type can be used, for example, to attach multiple digests calculated using different hashing algorithms in order to support a population of endpoints with different or evolving capabilities. Such an approach could support transitions away from weaker algorithms (see Section 6.6).

NOTE: '\ ' line wrapping per RFC 8792

```
Repr-Digest: \
  sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=;,\
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:
```



A recipient MAY ignore any or all digests. This allows the recipient to choose which hashing algorithm(s) to use for validation instead of verifying every digest.

A sender MAY send a digest without knowing whether the recipient supports a given hashing algorithm, or even knowing that the recipient will ignore it.

Content-Digest can be sent in a trailer section. In this case, Content-Digest MAY be merged into the header section; see Section 6.5.1 of [SEMANTICS].

#### 4. Integrity preference fields

Senders can indicate their interest in Integrity fields and hashing algorithm preferences using the Want-Repr-Digest or Want-Content-Digest fields. These can be used in both requests and responses.

Want-Repr-Digest indicates the sender's desire to receive a representation digest on messages associated with the request URI and representation metadata, using the Repr-Digest field.

Want-Content-Digest indicates the sender's desire to receive a content digest on messages associated with the request URI and representation metadata, using the Content-Digest field.

Want-Repr-Digest and Want-Content-Digest are Structured Fields Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]) where:

- \* keys convey the hashing algorithm (see Section 5);
- \* values MUST be of type Integer (Section 3.3.1 of [STRUCTURED-FIELDS]) in the range 0 to 10 inclusive. 1 is the least preferred, 10 is the most preferred, and a value of 0 means "not acceptable". Values convey an ascending, relative, weighted preference.

Want-Repr-Digest = sf-dictionary  
Want-Content-Digest = sf-dictionary

Examples:

Want-Repr-Digest: sha-256=1  
Want-Repr-Digest: sha-512=3, sha-256=10, unixsum=0  
Want-Content-Digest: sha-256=1  
Want-Content-Digest: sha-512=3, sha-256=10, unixsum=0

## 5. Hash Algorithms for HTTP Digest Fields Registry

The "Hash Algorithms for HTTP Digest Fields", maintained by IANA at <https://www.iana.org/assignments/http-dig-alg/> (<https://www.iana.org/assignments/http-dig-alg/>), registers algorithms for use with the Integrity and Integrity preference fields defined in this document.

This registry uses the Specification Required policy (Section 4.6 of [RFC8126]).

Registrations MUST include the following fields:

- \* **Algorithm Key:** the Structured Fields key value used in Repr-Digest, Content-Digest, Want-Repr-Digest or Want-Content-Digest field Dictionary member keys
- \* **Status:** the status of the algorithm. Use "standard" for standardized algorithms without known problems; "experimental" or some other appropriate value
  - e.g. according to the type and status of the primary document in which the algorithm is defined; "insecure" when the algorithm is insecure; "reserved" when the algorithm references a reserved token value
- \* **Description:** a short description of the algorithm
- \* **Reference(s):** a set of pointers to the primary documents defining the algorithm and key

Insecure hashing algorithms MAY be used to preserve integrity against corruption, but MUST NOT be used in a potentially adversarial setting; for example, when signing Integrity fields' values for authenticity.

The entries in Table 1 are registered by this document.

Algorithm Key	Status	Description	Reference(s)
sha-512	standard	The SHA-512 algorithm.	[RFC6234], [RFC4648], this document.
sha-256	standard	The SHA-256 algorithm.	[RFC6234], [RFC4648], this document.
md5	insecure	The MD5 algorithm. It is vulnerable to collision attacks; see [NO-MD5] and [CMU-836068]	[RFC1321], [RFC4648], this document.
sha	insecure	The SHA-1 algorithm. It is vulnerable to collision attacks; see [NO-SHA] and [IACR-2020-014]	[RFC3174], [RFC4648], [RFC6234] this document.
unixsum	insecure	The algorithm used by the UNIX "sum" command.	[RFC4648], [RFC6234], [UNIX], this document.
unixcksum	insecure	The algorithm used by the UNIX "cksum" command.	[RFC4648], [RFC6234], [UNIX], this document.
adler	insecure	The ADLER32 algorithm.	[RFC1950], this document.
crc32c	insecure	The CRC32c algorithm.	[RFC4960] appendix B, this document.

Table 1: Initial Hash Algorithms

## 6. Security Considerations

### 6.1. HTTP Messages Are Not Protected In Full

This document specifies a data integrity mechanism that protects HTTP "representation data" or content, but not HTTP header and trailer fields, from certain kinds of corruption.

Integrity fields are not intended to be a general protection against malicious tampering with HTTP messages. This can be achieved by combining it with other approaches such as transport-layer security or digital signatures.

### 6.2. End-to-End Integrity

Integrity fields can help detect "representation data" or content modification due to implementation errors, undesired "transforming proxies" (see Section 7.7 of [SEMANTICS]) or other actions as the data passes across multiple hops or system boundaries. Even a simple mechanism for end-to-end "representation data" integrity is valuable because a user agent can validate that resource retrieval succeeded before handing off to a HTML parser, video player etc. for parsing.

Note that using these mechanisms alone does not provide end-to-end integrity of HTTP messages over multiple hops, since metadata could be manipulated at any stage. Methods to protect metadata are discussed in Section 6.3.

### 6.3. Usage in Signatures

Digital signatures are widely used together with checksums to provide the certain identification of the origin of a message [NIST800-32]. Such signatures can protect one or more HTTP fields and there are additional considerations when Integrity fields are included in this set.

Digests explicitly depend on the "representation metadata" (e.g. the values of Content-Type, Content-Encoding etc). A signature that protects Integrity fields but not other "representation metadata" can expose the communication to tampering. For example, an actor could manipulate the Content-Type field-value and cause a digest validation failure at the recipient, preventing the application from accessing the representation. Such an attack consumes the resources of both endpoints. See also Section 2.2.

Signatures are likely to be deemed an adversarial setting when applying Integrity fields; see Section 5. Using signatures to protect the checksum of an empty representation allows receiving endpoints to detect if an eventual payload has been stripped or added.

Any mangling of Integrity fields, including digests' de-duplication or combining different field values (see Section 5.2 of [SEMANTICS]) might affect signature validation.

#### 6.4. Usage in Trailer Fields

Before sending Integrity fields in a trailer section, the sender should consider that intermediaries are explicitly allowed to drop any trailer (see Section 6.5.2 of [SEMANTICS]).

When Integrity fields are used in a trailer section, the field-values are received after the content. Eager processing of content before the trailer section prevents digest validation, possibly leading to processing of invalid data.

Not every hashing algorithm is suitable for use in the trailer section, some may require to pre-process the whole payload before sending a message (e.g. see [I-D.thomson-http-mice]).

#### 6.5. Usage with Encryption

The checksum of an encrypted payload can change between different messages depending on the encryption algorithm used; in those cases its value could not be used to provide a proof of integrity "at rest" unless the whole (e.g. encoded) content is persisted.

#### 6.6. Algorithm Agility

The security properties of hashing algorithms are not fixed. Algorithm Agility (see [RFC7696]) is achieved by providing implementations with flexibility to choose hashing algorithms from the IANA Hash Algorithms for HTTP Digest Fields registry; see Section 7.2.

The "standard" algorithms listed in this document are suitable for many purposes, including adversarial situations where hash functions might need to provide resistance to collision, first-preimage and second-preimage attacks. Algorithms listed as "insecure" either provide none of these properties, or are known to be weak (see [NO-MD5] and [NO-SHA]).

For adversarial situations, which of the "standard" algorithms are acceptable will depend on the level of protection the circumstances demand. As there is no negotiation, endpoints that depend on a digest for security will be vulnerable to attacks on the weakest algorithm they are willing to accept.

Transition from weak algorithms is supported by negotiation of hashing algorithm using Want-Repr-Digest or Want-Content-Digest (see Section 4) or by sending multiple digests from which the receiver chooses. Endpoints are advised that sending multiple values consumes resources, which may be wasted if the receiver ignores them (see Section 2).

While algorithm agility allows the migration to stronger algorithms it does not prevent the use of weaker algorithms. Integrity fields do not provide any mitigations for downgrade or substitution attacks (see Section 1 of [RFC6211]) of the hashing algorithm. To protect against such attacks, endpoints could restrict their set of supported algorithms to stronger ones and protect the fields value by using TLS and/or digital signatures.

#### 6.7. Resource exhaustion

Integrity fields validation consumes computational resources. In order to avoid resource exhaustion, implementations can restrict validation of the algorithm types, number of validations, or the size of content.

### 7. IANA Considerations

#### 7.1. HTTP Field Name Registration

IANA is asked to update the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]) according to the table below:

Field Name	Status	Reference
Repr-Digest	permanent	Section 2 of this document
Content-Digest	permanent	Section 3 of this document
Want-Repr-Digest	permanent	Section 4 of this document
Want-Content-Digest	permanent	Section 4 of this document
Digest	obsoleted	[RFC3230], Section 1.3 of this document
Want-Digest	obsoleted	[RFC3230], Section 1.3 of this document

Table 2

## 7.2. Establish the Hash Algorithms for HTTP Digest Fields Registry

This memo sets this specification to be the establishing document for the Hash Algorithms for HTTP Digest Fields (<https://www.iana.org/assignments/http-structured-dig-alg/>) registry defined in Section 5.

IANA is asked to initialize the registry with the entries in Table 1.

## 8. References

### 8.1. Normative References

[CMU-836068]

Carnegie Mellon University, Software Engineering Institute, "MD5 Vulnerable to collision attacks", 31 December 2008, <<https://www.kb.cert.org/vuls/id/836068/>>.

[IACR-2020-014]

Leurent, G. and T. Peyrin, "SHA-1 is a Shambles", 5 January 2020, <<https://eprint.iacr.org/2020/014.pdf>>.

[NIST800-32]

National Institute of Standards and Technology, U.S. Department of Commerce, "Introduction to Public Key Technology and the Federal PKI Infrastructure", February 2001, <<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-32.pdf>>.

- [RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <<https://www.rfc-editor.org/rfc/rfc1321>>.
- [RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <<https://www.rfc-editor.org/rfc/rfc1950>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC3174] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, DOI 10.17487/RFC3174, September 2001, <<https://www.rfc-editor.org/rfc/rfc3174>>.
- [RFC3230] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002, <<https://www.rfc-editor.org/rfc/rfc3230>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/rfc/rfc4960>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/rfc/rfc5234>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/rfc/rfc6234>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <<https://www.rfc-editor.org/rfc/rfc7405>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.



[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.

[SEMANTICS]

Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>>.

[STRUCTURED-FIELDS]

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

[UNIX] The Open Group, "The Single UNIX Specification, Version 2 - 6 Vol Set for UNIX 98", February 1997.

## 8.2. Informative References

[HTTP11] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-19>>.

[I-D.thomson-http-mice]

Thomson, M. and J. Yasskin, "Merkle Integrity Content Encoding", Work in Progress, Internet-Draft, draft-thomson-http-mice-03, 13 August 2018, <<https://datatracker.ietf.org/doc/html/draft-thomson-http-mice-03>>.

[NO-MD5] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <<https://www.rfc-editor.org/rfc/rfc6151>>.

[NO-SHA] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <<https://www.rfc-editor.org/rfc/rfc6194>>.

[PATCH] Dusséault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <<https://www.rfc-editor.org/rfc/rfc5789>>.

- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/rfc/rfc2818>>.
- [RFC6211] Schaad, J., "Cryptographic Message Syntax (CMS) Algorithm Identifier Protection Attribute", RFC 6211, DOI 10.17487/RFC6211, April 2011, <<https://www.rfc-editor.org/rfc/rfc6211>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <<https://www.rfc-editor.org/rfc/rfc7396>>.
- [RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <<https://www.rfc-editor.org/rfc/rfc7696>>.
- [RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <<https://www.rfc-editor.org/rfc/rfc7807>>.

#### Appendix A. Resource Representation and Representation Data

The following examples show how representation metadata, payload transformations and method impacts on the message and content. When the content contains non-printable characters (e.g. when it is compressed) it is shown as a Base64-encoded string.

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json

{"hello": "world"}
```

Figure 1: Request containing a JSON object without any content coding

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Encoding: gzip

H4sIAItWyFwC/6tWSlSyUlAypANQqgUAREcqfG0AAAA=
```

Figure 2: Request containing a gzip-encoded JSON object

Now the same content conveys a malformed JSON object, because the request does not indicate a content coding.

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json

H4sIAItWyFwC/6tWSlSyUlAypANQqgUAREcqfG0AAAA=
```

Figure 3: Request containing malformed JSON

A Range-Request alters the content, conveying a partial representation.

```
GET /entries/1234 HTTP/1.1
Host: foo.example
Range: bytes=1-7
```

Figure 4: Request for partial content

```
HTTP/1.1 206 Partial Content
Content-Encoding: gzip
Content-Type: application/json
Content-Range: bytes 1-7/18
```

```
iwgAla3RXA==
```

Figure 5: Partial response from a gzip-encoded representation

The method can also alter the content. For example, the response to a HEAD request does not carry content.

```
HEAD /entries/1234 HTTP/1.1
Host: foo.example
Accept: application/json
Accept-Encoding: gzip
```

Figure 6: HEAD request

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: gzip
```

Figure 7: Response to HEAD request (empty content)

Finally, the semantics of an HTTP response might decouple the effective request URI from the enclosed representation. In the example response below, the Content-Location header field indicates that the enclosed representation refers to the resource available at /authors/123, even though the request is directed to /authors/.

```
POST /authors/ HTTP/1.1
Host: foo.example
Accept: application/json
Content-Type: application/json

{"author": "Camilleri"}
```

Figure 8: POST request

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Location: /authors/123
Location: /authors/123

{"id": "123", "author": "Camilleri"}
```

Figure 9: Response with Content-Location header

## Appendix B. Examples of Unsolicited Digest

The following examples demonstrate interactions where a server responds with a Repr-Digest or Content-Digest fields even though the client did not solicit one using Want-Repr-Digest or Want-Content-Digest.

Some examples include JSON objects in the content. For presentation purposes, objects that fit completely within the line-length limits are presented on a single line using compact notation with no leading space. Objects that would exceed line-length limits are presented across multiple lines (one line per key-value pair) with 2 spaces of leading indentation.

Checksum mechanisms defined in this document are media-type agnostic and do not provide canonicalization algorithms for specific formats. Examples are calculated inclusive of any space. While examples can include both fields, Repr-Digest and Content-Digest can be returned independently.

### B.1. Server Returns Full Representation Data

In this example, the message content conveys complete representation data. This means that in the response, Repr-Digest and Content-Digest are both computed over the JSON object {"hello": "world"}, and thus have the same value.

```
GET /items/123 HTTP/1.1
Host: foo.example
```

Figure 10: GET request for an item

NOTE: '\ ' line wrapping per RFC 8792

```
HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: \
  sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
Content-Digest: \
  sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

Figure 11: Response with identical Repr-Digest and Content-Digest

### B.2. Server Returns No Representation Data

In this example, a HEAD request is used to retrieve the checksum of a resource.

The response Repr-Digest field-value is calculated over the JSON object {"hello": "world"}, which is not shown because there is no payload data. Content-Digest is computed on empty content.

```
HEAD /items/123 HTTP/1.1
Host: foo.example
```

Figure 12: HEAD request for an item

NOTE: '\ ' line wrapping per RFC 8792

```
HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: \
  sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
Content-Digest: \
  sha-256=:47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=:
```

Figure 13: Response with both Content-Digest and Digest; empty content

### B.3. Server Returns Partial Representation Data

In this example, the client makes a range request and the server responds with partial content.

```
GET /items/123 HTTP/1.1
Host: foo.example
Range: bytes=1-7
```

Figure 14: Request for partial content

NOTE: '\ ' line wrapping per RFC 8792

```
HTTP/1.1 206 Partial Content
Content-Type: application/json
Content-Range: bytes 1-7/18
Repr-Digest: \
  sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
Content-Digest: \
  sha-256=:Wqdirjg/u3J688ejbU1ApbjECpiUUtIwT8lY/z8lTno=:

"hello"
```

Figure 15: Partial response with both Content-Digest and Repr-Digest

In the response message above, note that the Repr-Digest and Content-Digests are different. The Repr-Digest field-value is calculated across the entire JSON object {"hello": "world"}, and the field is

```
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
```

However, since the message content is constrained to bytes 1-7, the Content-Digest field-value is calculated over the byte sequence "hello", thus resulting in

NOTE: '\ ' line wrapping per RFC 8792

```
Content-Digest: \
  sha-256=:Wqdirjg/u3J688ejbU1ApbjECpiUUtIwT8lY/z8lTno=:
```

#### B.4. Client and Server Provide Full Representation Data

The request contains a Repr-Digest field-value calculated on the enclosed representation. It also includes an Accept-Encoding: br header field that advertises the client supports Brotli encoding.

The response includes a Content-Encoding: br that indicates the selected representation is Brotli-encoded. The Repr-Digest field-value is therefore different compared to the request.

For presentation purposes, the response body is displayed as a Base64-encoded string because it contains non-printable characters.

```
PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept-Encoding: br
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

Figure 16: PUT Request with Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Location: /items/123
Content-Encoding: br
Content-Length: 22
Repr-Digest: sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=:

iwiAeyJoZWxsbyI6ICJ3b3JsZCJ9Aw==
```

Figure 17: Response with Digest of encoded response

#### B.5. Client Provides Full Representation Data, Server Provides No Representation Data

The request Repr-Digest field-value is calculated on the enclosed payload.

The response Repr-Digest field-value depends on the representation metadata header fields, including Content-Encoding: br even when the response does not contain content.

```
PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Length: 18
Accept-Encoding: br
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

```
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Encoding: br
Repr-Digest: sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=:
```

Figure 18: Empty response with Digest

## B.6. Client and Server Provide Full Representation Data

The response contains two digest values using different algorithms.

As the response body contains non-printable characters, it is displayed as a base64-encoded string.

```
PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept-Encoding: br
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

Figure 19: PUT Request with Digest

NOTE: ‘\’ line wrapping per RFC 8792

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: br
Content-Location: /items/123
Repr-Digest: \
  sha-256=:4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzigDZt3/h3Qxo=:\
  sha-512=:pxo7aYzcGI88pnDnoSmAnaOEVys0MABhgvHY9+VI+ElE60jBCwnMPyA/\
  s3NF3ZO5oIWA7lf8ukk+5KJzm3p5og==:

iwiAeyJoZWxsbyI6ICJ3b3JsZCJ9Aw==
```

Figure 20: Response with Digest of Encoded Content

## B.7. POST Response does not Reference the Request URI

The request Repr-Digest field-value is computed on the enclosed representation (see Section 2.1).

The representation enclosed in the response refers to the resource identified by Content-Location (see Section 6.4.2 of [SEMANTICS]). Repr-Digest is thus computed on the enclosed representation.



```
POST /books HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept: application/json
Accept-Encoding: identity
Repr-Digest: sha-256=:bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=:

{"title": "New Title"}
```

Figure 21: POST Request with Digest

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Location: /books/123
Location: /books/123
Repr-Digest: sha-256=:yxOAqEeoj+reqygSIsLpT0LhumrNkIds5uLKtmdLyYE=:

{
  "id": "123",
  "title": "New Title"
}
```

Figure 22: Response with Digest of Resource

Note that a 204 No Content response without content but with the same Repr-Digest field-value would have been legitimate too. In that case, Content-Digest would have been computed on an empty content.

#### B.8. POST Response Describes the Request Status

The request Repr-Digest field-value is computed on the enclosed representation (see Section 2.1).

The representation enclosed in the response describes the status of the request, so Repr-Digest is computed on that enclosed representation.

Response Repr-Digest has no explicit relation with the resource referenced by Location.

```
POST /books HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept: application/json
Accept-Encoding: identity
Repr-Digest: sha-256=:bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=:

{"title": "New Title"}
```

Figure 23: POST Request with Digest

```
HTTP/1.1 201 Created
Content-Type: application/json
Repr-Digest: sha-256=:2LBp5RKZGpsSNf8BPXlXrX4Td4Tf5R5bZ9z7kdi5VvY=:
Location: /books/123

{
  "status": "created",
  "id": "123",
  "ts": 1569327729,
  "instance": "/books/123"
}
```

Figure 24: Response with Digest of Representation

#### B.9. Digest with PATCH

This case is analogous to a POST request where the target resource reflects the effective request URI.

The PATCH request uses the `application/merge-patch+json` media type defined in [RFC7396].

Repr-Digest is calculated on the enclosed payload, which corresponds to the patch document.

The response Repr-Digest field-value is computed on the complete representation of the patched resource.

```
PATCH /books/123 HTTP/1.1
Host: foo.example
Content-Type: application/merge-patch+json
Accept: application/json
Accept-Encoding: identity
Repr-Digest: sha-256=:bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=:

{"title": "New Title"}
```

Figure 25: PATCH Request with Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: sha-256=:yx0AqEeoj+reqygSIsLpT0LhumrNkIds5uLKtmdLyYE=:

{
  "id": "123",
  "title": "New Title"
}
```

Figure 26: Response with Digest of Representation

Note that a 204 No Content response without content but with the same Repr-Digest field-value would have been legitimate too.

#### B.10. Error responses

In error responses, the "representation data" does not necessarily refer to the target resource. Instead, it refers to the representation of the error.

In the following example, a client sends the same request from Figure 25 to patch the resource located at /books/123. However, the resource does not exist and the server generates a 404 response with a body that describes the error in accordance with [RFC7807].

The response Repr-Digest field-value is computed on this enclosed representation.

```
HTTP/1.1 404 Not Found
Content-Type: application/problem+json
Repr-Digest: sha-256=:KPqhVXAT25LLitVlw00l67unHmVQusu+fpxm65zAsvk=:

{
  "title": "Not Found",
  "detail": "Cannot PATCH a non-existent resource",
  "status": 404
}
```

Figure 27: Response with Digest of Error Representation

#### B.11. Use with Trailer Fields and Transfer Coding

An origin server sends Repr-Digest as trailer field, so it can calculate digest-value while streaming content and thus mitigate resource consumption. The Repr-Digest field-value is the same as in Appendix B.1 because Repr-Digest is designed to be independent from the use of one or more transfer codings (see Section 2).

```
GET /items/123 HTTP/1.1
Host: foo.example
```

Figure 28: GET Request

```
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
Trailer: Digest

8\r\n
{"hello"\r\n
8
: "world\r\n
2\r\n
"}\r\n
0\r\n
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:
```

Figure 29: Chunked Response with Digest

#### Appendix C. Examples of Want-Repr-Digest Solicited Digest

The following examples demonstrate interactions where a client solicits a Repr-Digest using Want-Repr-Digest. The behavior of Content-Digest and Want-Content-Digest is identical.

Some examples include JSON objects in the content. For presentation purposes, objects that fit completely within the line-length limits are presented on a single line using compact notation with no leading space. Objects that would exceed line-length limits are presented across multiple lines (one line per key-value pair) with 2 spaced of leading indentation.

Checksum mechanisms described in this document are media-type agnostic and do not provide canonicalization algorithms for specific formats. Examples are calculated inclusive of any space.

##### C.1. Server Selects Client's Least Preferred Algorithm

The client requests a digest, preferring "sha". The server is free to reply with "sha-256" anyway.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Repr-Digest: sha-256=3, sha=10
```

Figure 30: GET Request with Want-Repr-Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

{"hello": "world"}
```

Figure 31: Response with Different Algorithm

### C.2. Server Selects Algorithm Unsupported by Client

The client requests a "sha" digest because that is the only algorithm it supports. The server is not obliged to produce a response containing a "sha" digest, it instead uses a different algorithm.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Repr-Digest: sha=10
```

Figure 32: GET Request with Want-Repr-Digest

NOTE: ‘\’ line wrapping per RFC 8792

```
HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: \
  sha-512=:WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+AbwAgBWnrI\
  iYllu7BNNyealdVLvRwEmTHWXvJwew==:

{"hello": "world"}
```

Figure 33: Response with Unsupported Algorithm

### C.3. Server Does Not Support Client Algorithm and Returns an Error

Appendix C.2 is an example where a server ignores the client's preferred digest algorithm. Alternatively a server can also reject the request and return an error.

In this example, the client requests a "sha" Repr-Digest, and the server returns an error with problem details [RFC7807] contained in the content. The problem details contain a list of the hashing algorithms that the server supports. This is purely an example, this specification does not define any format or requirements for such content.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Repr-Digest: sha=10
```

Figure 34: GET Request with Want-Repr-Digest

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{
  "title": "Bad Request",
  "detail": "Supported hashing algorithms: sha-256, sha-512",
  "status": 400
}
```

Figure 35: Response advertising the supported algorithms

#### Acknowledgements

This document is based on ideas from [RFC3230], so thanks to J. Mogul and A. Van Hoff for their great work. The original idea of refreshing RFC3230 arose from an interesting discussion with M. Nottingham, J. Yasskin and M. Thomson when reviewing the MICE content coding.

Thanks to Julian Reschke for his valuable contributions to this document, and to the following contributors that have helped improve this specification by reporting bugs, asking smart questions, drafting or reviewing text, and evaluating open issues: Mike Bishop, Brian Campbell, Matthew Kerwin, James Manger, Tommy Pauly, Sean Turner, Justin Richer, and Erik Wilde.

#### Code Samples

\_RFC Editor: Please remove this section before publication.\_

How can I generate and validate the Repr-Digest values shown in the examples throughout this document?

The following python3 code can be used to generate digests for JSON objects using SHA algorithms for a range of encodings. Note that these are formatted as base64. This function could be adapted to other algorithms and should take into account their specific formatting rules.

```

import base64, json, hashlib, brotli, logging
log = logging.getLogger()

def encode_item(item, encoding=lambda x: x):
    indent = 2 if isinstance(item, dict) and len(item) > 1 else None
    json_bytes = json.dumps(item, indent=indent).encode()
    return encoding(json_bytes)

def digest_bytes(bytes_, algorithm=hashlib.sha256):
    checksum_bytes = algorithm(bytes_).digest()
    log.warning("Log bytes: \n[%r]", bytes_)
    return base64.encodebytes(checksum_bytes).strip()

def digest(item, encoding=lambda x: x, algorithm=hashlib.sha256):
    content_encoded = encode_item(item, encoding)
    return digest_bytes(content_encoded, algorithm)

item = {"hello": "world"}

print("Encoding | hashing algorithm | digest-value")
print("Identity | sha256 |", digest(item))
# Encoding | hashing algorithm | digest-value
# Identity | sha256 | X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

print("Encoding | hashing algorithm | digest-value")
print("Brotli | sha256 |", digest(item, encoding=brotli.compress))
# Encoding | hashing algorithm | digest-value
# Brotli | sha256 | 4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=

print("Encoding | hashing algorithm | digest-value")
print("Identity | sha512 |", digest(item, algorithm=hashlib.sha512))
print("Brotli | sha512 |", digest(item, algorithm=hashlib.sha512,
                                encoding=brotli.compress))
# Encoding | hashing algorithm | digest-value
# Identity | sha512 | b'WZDPaVn/7XgHaAy8pmojAkGWRx2UFChF41A2svX+TaPm'
#                      '+AbwAgBWnrIiYllu7BNNyealdVLvRwEmTHWXvJwew=='
# Brotli | sha512 | b'pxo7aYzcGI88pnDnoSmAnaOEVys0MABhgvHY9+VI+E1E6'
#                      '0jBCwnMPyA/s3NF3ZO5oIWA7lf8ukk+5KJzm3p5og=='

```

#### Changes

\_RFC Editor: Please remove this section before publication.\_

Since draft-ietf-httpbis-digest-headers-06

- \* Remove id-sha-256 and id-sha-512 from the list of supported algorithms #855

Since draft-ietf-httpbis-digest-headers-05

- \* Reboot digest-algorithm values registry #1567
- \* Add Content-Digest #1542
- \* Remove SRI section #1478

Since draft-ietf-httpbis-digest-headers-04

- \* Improve SRI section #1354
- \* About duplicate digest-algorithms #1221
- \* Improve security considerations #852
- \* md5 and sha deprecation references #1392
- \* Obsolete 3230 #1395
- \* Editorial #1362

Since draft-ietf-httpbis-digest-headers-03

- \* Reference semantics-12
- \* Detail encryption quirks
- \* Details on Algorithm agility #1250
- \* Obsolete parameters #850

Since draft-ietf-httpbis-digest-headers-02

- \* Deprecate SHA-1 #1154
- \* Avoid id-\* with encrypted content
- \* Digest is independent from MESSAGING and HTTP/1.1 is not normative #1215
- \* Identity is not a valid field value for content-encoding #1223
- \* Mention trailers #1157
- \* Reference httpbis-semantics #1156
- \* Add contentMD5 as an obsoleted digest-algorithm #1249



- \* Use lowercase digest-algorithms names in the doc and in the digest-algorithm IANA table.

Since draft-ietf-httpbis-digest-headers-01

- \* Digest of error responses is computed on the error representation-data #1004
- \* Effect of HTTP semantics on payload and message body moved to appendix #1122
- \* Editorial refactoring, moving headers sections up. #1109-#1112, #1116, #1117, #1122-#1124

Since draft-ietf-httpbis-digest-headers-00

- \* Align title with document name
- \* Add id-sha-\* algorithm examples #880
- \* Reference [RFC6234] and [RFC3174] instead of FIPS-1
- \* Deprecate MD5
- \* Obsolete ADLER-32 but don't forbid it #828
- \* Update CRC32C value in IANA table #828
- \* Use when acting on resources (POST, PATCH) #853
- \* Added Relationship with SRI, draft Use Cases #868, #971
- \* Warn about the implications of Content-Location

#### Authors' Addresses

Roberto Polli  
Team Digitale, Italian Government  
Italy  
Email: robipolli@gmail.com

Lucas Pardue  
Cloudflare  
Email: lucaspardue.24.7@gmail.com

HTTP  
Internet-Draft  
Intended status: Standards Track  
Expires: December 5, 2020

M. Nottingham  
Fastly  
P-H. Kamp  
The Varnish Cache Project  
June 3, 2020

Structured Field Values for HTTP  
draft-ietf-httpbis-header-structure-19

Abstract

This document describes a set of data types and associated algorithms that are intended to make it easier and safer to define and handle HTTP header and trailer fields, known as "Structured Fields", "Structured Headers", or "Structured Trailers". It is intended for use by specifications of new HTTP fields that wish to use a common syntax that is more restrictive than traditional HTTP field values.

Note to Readers

\_RFC EDITOR: please remove this section before publication\_

Discussion of this draft takes place on the HTTP working group mailing list ([ietf-http-wg@w3.org](mailto:ietf-http-wg@w3.org)), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/> [1].

Working Group information can be found at <https://httpwg.github.io/> [2]; source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions/labels/header-structure> [3].

Tests for implementations are collected at <https://github.com/httpwg/structured-field-tests> [4].

Implementations are tracked at <https://github.com/httpwg/wiki/wiki/Structured-Headers> [5].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 5, 2020.

#### Copyright Notice

Copyright (c) 2020 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 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
1.1. Intentionally Strict Processing . . . . .	4
1.2. Notational Conventions . . . . .	5
2. Defining New Structured Fields . . . . .	5
3. Structured Data Types . . . . .	8
3.1. Lists . . . . .	9
3.1.1. Inner Lists . . . . .	9
3.1.2. Parameters . . . . .	10
3.2. Dictionaries . . . . .	11
3.3. Items . . . . .	12
3.3.1. Integers . . . . .	13
3.3.2. Decimals . . . . .	13
3.3.3. Strings . . . . .	14
3.3.4. Tokens . . . . .	15
3.3.5. Byte Sequences . . . . .	15
3.3.6. Booleans . . . . .	15
4. Working With Structured Fields in HTTP . . . . .	16
4.1. Serializing Structured Fields . . . . .	16
4.1.1. Serializing a List . . . . .	16
4.1.2. Serializing a Dictionary . . . . .	18
4.1.3. Serializing an Item . . . . .	19
4.1.4. Serializing an Integer . . . . .	20
4.1.5. Serializing a Decimal . . . . .	20
4.1.6. Serializing a String . . . . .	21

4.1.7. Serializing a Token . . . . .	22
4.1.8. Serializing a Byte Sequence . . . . .	22
4.1.9. Serializing a Boolean . . . . .	22
4.2. Parsing Structured Fields . . . . .	23
4.2.1. Parsing a List . . . . .	24
4.2.2. Parsing a Dictionary . . . . .	26
4.2.3. Parsing an Item . . . . .	27
4.2.4. Parsing an Integer or Decimal . . . . .	29
4.2.5. Parsing a String . . . . .	30
4.2.6. Parsing a Token . . . . .	31
4.2.7. Parsing a Byte Sequence . . . . .	32
4.2.8. Parsing a Boolean . . . . .	33
5. IANA Considerations . . . . .	33
6. Security Considerations . . . . .	33
7. References . . . . .	33
7.1. Normative References . . . . .	33
7.2. Informative References . . . . .	34
7.3. URIs . . . . .	35
Appendix A. Frequently Asked Questions . . . . .	35
A.1. Why not JSON? . . . . .	35
Appendix B. Implementation Notes . . . . .	36
Appendix C. Changes . . . . .	36
C.1. Since draft-ietf-httpbis-header-structure-18 . . . . .	37
C.2. Since draft-ietf-httpbis-header-structure-17 . . . . .	37
C.3. Since draft-ietf-httpbis-header-structure-16 . . . . .	37
C.4. Since draft-ietf-httpbis-header-structure-15 . . . . .	37
C.5. Since draft-ietf-httpbis-header-structure-14 . . . . .	38
C.6. Since draft-ietf-httpbis-header-structure-13 . . . . .	38
C.7. Since draft-ietf-httpbis-header-structure-12 . . . . .	39
C.8. Since draft-ietf-httpbis-header-structure-11 . . . . .	39
C.9. Since draft-ietf-httpbis-header-structure-10 . . . . .	39
C.10. Since draft-ietf-httpbis-header-structure-09 . . . . .	39
C.11. Since draft-ietf-httpbis-header-structure-08 . . . . .	40
C.12. Since draft-ietf-httpbis-header-structure-07 . . . . .	40
C.13. Since draft-ietf-httpbis-header-structure-06 . . . . .	41
C.14. Since draft-ietf-httpbis-header-structure-05 . . . . .	41
C.15. Since draft-ietf-httpbis-header-structure-04 . . . . .	41
C.16. Since draft-ietf-httpbis-header-structure-03 . . . . .	41
C.17. Since draft-ietf-httpbis-header-structure-02 . . . . .	41
C.18. Since draft-ietf-httpbis-header-structure-01 . . . . .	42
C.19. Since draft-ietf-httpbis-header-structure-00 . . . . .	42
Acknowledgements . . . . .	42
Authors' Addresses . . . . .	42

## 1. Introduction

Specifying the syntax of new HTTP header (and trailer) fields is an onerous task; even with the guidance in Section 8.3.1 of [RFC7231], there are many decisions - and pitfalls - for a prospective HTTP field author.

Once a field is defined, bespoke parsers and serializers often need to be written, because each field value has slightly different handling of what looks like common syntax.

This document introduces a set of common data structures for use in definitions of new HTTP field values to address these problems. In particular, it defines a generic, abstract model for them, along with a concrete serialization for expressing that model in HTTP [RFC7230] header and trailer fields.

A HTTP field that is defined as a "Structured Header" or "Structured Trailer" (if the field can be either, it is a "Structured Field") uses the types defined in this specification to define its syntax and basic handling rules, thereby simplifying both its definition by specification writers and handling by implementations.

Additionally, future versions of HTTP can define alternative serializations of the abstract model of these structures, allowing fields that use that model to be transmitted more efficiently without being redefined.

Note that it is not a goal of this document to redefine the syntax of existing HTTP fields; the mechanisms described herein are only intended to be used with fields that explicitly opt into them.

Section 2 describes how to specify a Structured Field.

Section 3 defines a number of abstract data types that can be used in Structured Fields.

Those abstract types can be serialized into and parsed from HTTP field values using the algorithms described in Section 4.

### 1.1. Intentionally Strict Processing

This specification intentionally defines strict parsing and serialization behaviors using step-by-step algorithms; the only error handling defined is to fail the operation altogether.

It is designed to encourage faithful implementation and therefore good interoperability. Therefore, an implementation that tried to be

helpful by being more tolerant of input would make interoperability worse, since that would create pressure on other implementations to implement similar (but likely subtly different) workarounds.

In other words, strict processing is an intentional feature of this specification; it allows non-conformant input to be discovered and corrected by the producer early, and avoids both interoperability and security issues that might otherwise result.

Note that as a result of this strictness, if a field is appended to by multiple parties (e.g., intermediaries, or different components in the sender), an error in one party's value is likely to cause the entire field value to fail parsing.

## 1.2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses algorithms to specify parsing and serialization behaviors, and the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] to illustrate expected syntax in HTTP header fields. In doing so, it uses the VCHAR, SP, DIGIT, ALPHA and DQUOTE rules from [RFC5234]. It also includes the tchar and OWS rules from [RFC7230].

When parsing from HTTP fields, implementations MUST have behavior that is indistinguishable from following the algorithms. If there is disagreement between the parsing algorithms and ABNF, the specified algorithms take precedence.

For serialization to HTTP fields, the ABNF illustrates their expected wire representations, and the algorithms define the recommended way to produce them. Implementations MAY vary from the specified behavior so long as the output is still correctly handled by the parsing algorithm.

## 2. Defining New Structured Fields

To specify a HTTP field as a Structured Field, its authors needs to:

- o Normatively reference this specification. Recipients and generators of the field need to know that the requirements of this document are in effect.

- o Identify whether the field is a Structured Header (i.e., it can only be used in the header section - the common case), a Structured Trailer (only in the trailer section), or a Structured Field (both).
- o Specify the type of the field value; either List (Section 3.1), Dictionary (Section 3.2), or Item (Section 3.3).
- o Define the semantics of the field value.
- o Specify any additional constraints upon the field value, as well as the consequences when those constraints are violated.

Typically, this means that a field definition will specify the top-level type - List, Dictionary or Item - and then define its allowable types, and constraints upon them. For example, a header defined as a List might have all Integer members, or a mix of types; a header defined as an Item might allow only Strings, and additionally only strings beginning with the letter "Q", or strings in lowercase. Likewise, Inner Lists (Section 3.1.1) are only valid when a field definition explicitly allows them.

When parsing fails, the entire field is ignored (see Section 4.2); in most situations, violating field-specific constraints should have the same effect. Thus, if a header is defined as an Item and required to be an Integer, but a String is received, the field will by default be ignored. If the field requires different error handling, this should be explicitly specified.

Both Items and Inner Lists allow parameters as an extensibility mechanism; this means that values can later be extended to accommodate more information, if need be. To preserve forward compatibility, field specifications are discouraged from defining the presence of an unrecognized Parameter as an error condition.

To further assure that this extensibility is available in the future, and to encourage consumers to use a complete parser implementation, a field definition can specify that "grease" Parameters be added by senders. A specification could stipulate that all Parameters that fit a defined pattern are reserved for this use and then encourage them to be sent on some portion of requests. This helps to discourage recipients from writing a parser that does not account for Parameters.

Specifications that use Dictionaries can also allow for forward compatibility by requiring that the presence of - as well as value and type associated with - unknown members be ignored. Later

specifications can then add additional members, specifying constraints on them as appropriate.

An extension to a structured field can then require that an entire field value be ignored by a recipient that understands the extension if constraints on the value it defines are not met.

A field definition cannot relax the requirements of this specification because doing so would preclude handling by generic software; they can only add additional constraints (for example, on the numeric range of Integers and Decimals, the format of Strings and Tokens, the types allowed in a Dictionary's values, or the number of Items in a List). Likewise, field definitions can only use this specification for the entire field value, not a portion thereof.

This specification defines minimums for the length or number of various structures supported by implementations. It does not specify maximum sizes in most cases, but authors should be aware that HTTP implementations do impose various limits on the size of individual fields, the total number of fields, and/or the size of the entire header or trailer section.

Specifications can refer to a field name as a "structured header name", "structured trailer name" or "structured field name" as appropriate. Likewise, they can refer its field value as a "structured header value", "structured trailer value" or "structured field value" as necessary. Field definitions are encouraged to use the ABNF rules beginning with "sf-" defined in this specification; other rules in this specification are not intended for their use.

For example, a fictitious Foo-Example header field might be specified as:



--8<--

#### 42. Foo-Example Header

The Foo-Example HTTP header field conveys information about how much Foo the message has.

Foo-Example is a Item Structured Header [RFCxxxx]. Its value MUST be an Integer (Section Y.Y of [RFCxxxx]). Its ABNF is:

```
Foo-Example = sf-integer
```

Its value indicates the amount of Foo in the message, and MUST be between 0 and 10, inclusive; other values MUST cause the entire header field to be ignored.

The following parameters are defined:

- \* A Parameter whose name is "foourl", and whose value is a String (Section Y.Y of [RFCxxxx]), conveying the Foo URL for the message. See below for processing requirements.

"foourl" contains a URI-reference (Section 4.1 of [RFC3986]). If its value is not a valid URI-reference, the entire header field MUST be ignored. If its value is a relative reference (Section 4.2 of [RFC3986]), it MUST be resolved (Section 5 of [RFC3986]) before being used.

For example:

```
Foo-Example: 2; foourl="https://foo.example.com/"
-->8--
```

### 3. Structured Data Types

This section defines the abstract types for Structured Fields. The ABNF provided represents the on-wire format in HTTP field values.

In summary:

- o There are three top-level types that a HTTP field can be defined as: Lists, Dictionaries, and Items.
- o Lists and Dictionaries are containers; their members can be Items or Inner Lists (which are themselves arrays of Items).
- o Both Items and Inner Lists can be parameterized with key/value pairs.

### 3.1. Lists

Lists are arrays of zero or more members, each of which can be an Item (Section 3.3) or an Inner List (Section 3.1.1), both of which can be Parameterized (Section 3.1.2).

The ABNF for Lists in HTTP fields is:

```
sf-list      = list-member *( OWS "," OWS list-member )
list-member  = sf-item / inner-list
```

Each member is separated by a comma and optional whitespace. For example, a field whose value is defined as a List of Strings could look like:

```
Example-StrList: "foo", "bar", "It was the best of times."
```

An empty List is denoted by not serializing the field at all. This implies that fields defined as Lists have a default empty value.

Note that Lists can have their members split across multiple lines inside a header or trailer section, as per Section 3.2.2 of [RFC7230]; for example, the following are equivalent:

```
Example-Hdr: foo, bar
```

and

```
Example-Hdr: foo
Example-Hdr: bar
```

However, individual members of a List cannot be safely split between across lines; see Section 4.2 for details.

Parsers MUST support Lists containing at least 1024 members. Field specifications can constrain the types and cardinality of individual List values as they require.

#### 3.1.1. Inner Lists

An Inner List is an array of zero or more Items (Section 3.3). Both the individual Items and the Inner List itself can be Parameterized (Section 3.1.2).

The ABNF for Inner Lists is:

```
inner-list   = "(" *SP [ sf-item *( 1*SP sf-item ) *SP ] ")"
              parameters
```

Inner Lists are denoted by surrounding parenthesis, and have their values delimited by one or more spaces. A field whose value is defined as a List of Inner Lists of Strings could look like:

Example-StrListList: ("foo" "bar"), ("baz"), ("bat" "one"), ()

Note that the last member in this example is an empty Inner List.

A header field whose value is defined as a List of Inner Lists with Parameters at both levels could look like:

Example-ListListParam: ("foo"; a=1;b=2);lvl=5, ("bar" "baz");lvl=1

Parsers MUST support Inner Lists containing at least 256 members. Field specifications can constrain the types and cardinality of individual Inner List members as they require.

### 3.1.2. Parameters

Parameters are an ordered map of key-value pairs that are associated with an Item (Section 3.3) or Inner List (Section 3.1.1). The keys are unique within the scope the Parameters they occur within, and the values are bare items (i.e., they themselves cannot be parameterized; see Section 3.3).

The ABNF for Parameters is:

```
parameters    = *( ";" *SP parameter )
parameter     = param-name [ "=" param-value ]
param-name    = key
key           = ( lcalpha / "*" )
              *( lcalpha / DIGIT / "_" / "-" / "." / "*" )
lcalpha       = %x61-7A ; a-z
param-value   = bare-item
```

Note that Parameters are ordered as serialized, and Parameter keys cannot contain uppercase letters. A parameter is separated from its Item or Inner List and other parameters by a semicolon. For example:

Example-ParamList: abc;a=1;b=2; cde\_456, (ghi;jk=4 l);q="9";r=w

Parameters whose value is Boolean (see Section 3.3.6) true MUST omit that value when serialized. For example, the "a" parameter here is true, while the "b" parameter is false:

Example-Int: 1; a; b=?0

Note that this requirement is only on serialization; parsers are still required to correctly handle the true value when it appears in a parameter.

Parsers MUST support at least 256 parameters on an Item or Inner List, and support parameter keys with at least 64 characters. Field specifications can constrain the order of individual Parameters, as well as their values' types as required.

### 3.2. Dictionaries

Dictionaries are ordered maps of name-value pairs, where the names are short textual strings and the values are Items (Section 3.3) or arrays of Items, both of which can be Parameterized (Section 3.1.2). There can be zero or more members, and their names are unique in the scope of the Dictionary they occur within.

Implementations MUST provide access to Dictionaries both by index and by name. Specifications MAY use either means of accessing the members.

The ABNF for Dictionaries is:

```
sf-dictionary = dict-member *( OWS "," OWS dict-member )
dict-member   = member-name [ "=" member-value ]
member-name    = key
member-value   = sf-item / inner-list
```

Members are ordered as serialized, and separated by a comma with optional whitespace. Member names cannot contain uppercase characters. Names and values are separated by "=" (without whitespace). For example:

Example-Dict: en="Applepie", da=:w4ZibGV0w6ZydGU=:

Note that in this example, the final "=" is due to the inclusion of a Byte Sequence; see Section 3.3.5.

Members whose value is Boolean (see Section 3.3.6) true MUST omit that value when serialized. For example, here both "b" and "c" are true:

Example-Dict: a=?0, b, c; foo=bar

Note that this requirement is only on serialization; parsers are still required to correctly handle the true Boolean value when it appears in Dictionary values.

A Dictionary with a member whose value is an Inner List of Tokens:

Example-DictList: rating=1.5, feelings=(joy sadness)

A Dictionary with a mix of Items and Inner Lists, some with Parameters:

Example-MixDict: a=(1 2), b=3, c=4;aa=bb, d=(5 6);valid

As with lists, an empty Dictionary is represented by omitting the entire field. This implies that fields defined as Dictionaries have a default empty value.

Typically, a field specification will define the semantics of Dictionaries by specifying the allowed type(s) for individual members by their names, as well as whether their presence is required or optional. Recipients MUST ignore names that are undefined or unknown, unless the field's specification specifically disallows them.

Note that Dictionaries can have their members split across multiple lines inside a header or trailer section; for example, the following are equivalent:

Example-Hdr: foo=1, bar=2

and

Example-Hdr: foo=1

Example-Hdr: bar=2

However, individual members of a Dictionary cannot be safely split between lines; see Section 4.2 for details.

Parsers MUST support Dictionaries containing at least 1024 name/value pairs, and names with at least 64 characters. Field specifications can constrain the order of individual Dictionary members, as well as their values' types as required.

### 3.3. Items

An Item can be a Integer (Section 3.3.1), Decimal (Section 3.3.2), String (Section 3.3.3), Token (Section 3.3.4), Byte Sequence (Section 3.3.5), or Boolean (Section 3.3.6). It can have associated Parameters (Section 3.1.2).

The ABNF for Items is:

```
sf-item    = bare-item parameters
bare-item  = sf-integer / sf-decimal / sf-string / sf-token
           / sf-binary / sf-boolean
```

For example, a header field that is defined to be an Item that is an Integer might look like:

Example-IntItemHeader: 5

or with Parameters:

Example-IntItem: 5; foo=bar

### 3.3.1. Integers

Integers have a range of -999,999,999,999,999 to 999,999,999,999,999 inclusive (i.e., up to fifteen digits, signed), for IEEE 754 compatibility ([IEEE754]).

The ABNF for Integers is:

```
sf-integer = ["-"] 1*15DIGIT
```

For example:

Example-Integer: 42

Integers larger than 15 digits can be supported in a variety of ways; for example, by using a String (Section 3.3.3), Byte Sequence (Section 3.3.5), or a parameter on an Integer that acts as a scaling factor.

While it is possible to serialise Integers with leading zeros (e.g., "0002", "-01") and signed zero ("-0"), these distinctions may not be preserved by implementations.

Note that commas in Integers are used in this section's prose only for readability; they are not valid in the wire format.

### 3.3.2. Decimals

Decimals are numbers with an integer and a fractional component. The integer component has at most 12 digits; the fractional component has at most three digits.

The ABNF for decimals is:

```
sf-decimal = ["-"] 1*12DIGIT "." 1*3DIGIT
```

For example, a header whose value is defined as a Decimal could look like:

Example-Decimal: 4.5

While it is possible to serialise Decimals with leading zeros (e.g., "0002.5", "-01.334"), trailing zeros (e.g., "5.230", "-0.40"), and signed zero (e.g., "-0.0"), these distinctions may not be preserved by implementations.

Note that the serialisation algorithm (Section 4.1.5) rounds input with more than three digits of precision in the fractional component. If an alternative rounding strategy is desired, this should be specified by the header definition to occur before serialisation.

### 3.3.3. Strings

Strings are zero or more printable ASCII [RFC0020] characters (i.e., the range %x20 to %x7E). Note that this excludes tabs, newlines, carriage returns, etc.

The ABNF for Strings is:

```
sf-string = DQUOTE *chr DQUOTE
chr       = unescaped / escaped
unescaped = %x20-21 / %x23-5B / %x5D-7E
escaped   = "\" ( DQUOTE / "\" )
```

Strings are delimited with double quotes, using a backslash ("\") to escape double quotes and backslashes. For example:

Example-String: "hello world"

Note that Strings only use DQUOTE as a delimiter; single quotes do not delimit Strings. Furthermore, only DQUOTE and "\"" can be escaped; other characters after "\"" MUST cause parsing to fail.

Unicode is not directly supported in Strings, because it causes a number of interoperability issues, and - with few exceptions - field values do not require it.

When it is necessary for a field value to convey non-ASCII content, a Byte Sequence (Section 3.3.5) can be specified, along with a character encoding (preferably [UTF-8]).

Parsers MUST support Strings (after any decoding) with at least 1024 characters.

### 3.3.4. Tokens

Tokens are short textual words; their abstract model is identical to their expression in the HTTP field value serialization.

The ABNF for Tokens is:

```
sf-token = ( ALPHA / "*" ) *( tchar / ":" / "/" )
```

For example:

Example-Token: fool23/456

Parsers MUST support Tokens with at least 512 characters.

Note that Token allows the same characters as the "token" ABNF rule defined in [RFC7230], with the exceptions that the first character is required to be either ALPHA or "\*", and ":" and "/" are also allowed in subsequent characters.

### 3.3.5. Byte Sequences

Byte Sequences can be conveyed in Structured Fields.

The ABNF for a Byte Sequence is:

```
sf-binary = ":" *(base64) ":"  
base64    = ALPHA / DIGIT / "+" / "/" / "="
```

A Byte Sequence is delimited with colons and encoded using base64 ([RFC4648], Section 4). For example:

Example-Binary: :cHJldGVuZCB0aGZlZIGlZIGJpbmFyeSBjb250ZW50Lg==:

Parsers MUST support Byte Sequences with at least 16384 octets after decoding.

### 3.3.6. Booleans

Boolean values can be conveyed in Structured Fields.

The ABNF for a Boolean is:

```
sf-boolean = "?" boolean  
boolean    = "0" / "1"
```

A Boolean is indicated with a leading "?" character followed by a "1" for a true value or "0" for false. For example:



Example-Bool: ?1

Note that in Dictionary (Section 3.2) and Parameter (Section 3.1.2) values, Boolean true is indicated by omitting the value.

#### 4. Working With Structured Fields in HTTP

This section defines how to serialize and parse Structured Fields in textual HTTP field values and other encodings compatible with them (e.g., in HTTP/2 [RFC7540] before compression with HPACK [RFC7541]).

##### 4.1. Serializing Structured Fields

Given a structure defined in this specification, return an ASCII string suitable for use in a HTTP field value.

1. If the structure is a Dictionary or List and its value is empty (i.e., it has no members), do not serialize the field at all (i.e., omit both the field-name and field-value).
2. If the structure is a List, let `output_string` be the result of running Serializing a List (Section 4.1.1) with the structure.
3. Else if the structure is a Dictionary, let `output_string` be the result of running Serializing a Dictionary (Section 4.1.2) with the structure.
4. Else if the structure is an Item, let `output_string` be the result of running Serializing an Item (Section 4.1.3) with the structure.
5. Else, fail serialization.
6. Return `output_string` converted into an array of bytes, using ASCII encoding [RFC0020].

###### 4.1.1. Serializing a List

Given an array of (member\_value, parameters) tuples as `input_list`, return an ASCII string suitable for use in a HTTP field value.

1. Let `output` be an empty string.
2. For each (member\_value, parameters) of `input_list`:
  1. If `member_value` is an array, append the result of running Serializing an Inner List (Section 4.1.1.1) with (member\_value, parameters) to `output`.

2. Otherwise, append the result of running Serializing an Item (Section 4.1.3) with (member\_value, parameters) to output.
3. If more member\_values remain in input\_list:
  1. Append ",", to output.
  2. Append a single SP to output.
3. Return output.

#### 4.1.1.1. Serializing an Inner List

Given an array of (member\_value, parameters) tuples as inner\_list, and parameters as list\_parameters, return an ASCII string suitable for use in a HTTP field value.

1. Let output be the string "(".
2. For each (member\_value, parameters) of inner\_list:
  1. Append the result of running Serializing an Item (Section 4.1.3) with (member\_value, parameters) to output.
  2. If more values remain in inner\_list, append a single SP to output.
3. Append ")" to output.
4. Append the result of running Serializing Parameters (Section 4.1.1.2) with list\_parameters to output.
5. Return output.

#### 4.1.1.2. Serializing Parameters

Given an ordered Dictionary as input\_parameters (each member having a param\_name and a param\_value), return an ASCII string suitable for use in a HTTP field value.

1. Let output be an empty string.
2. For each param\_name with a value of param\_value in input\_parameters:
  1. Append ";" to output.

2. Append the result of running Serializing a Key (Section 4.1.1.3) with `param_name` to output.
3. If `param_value` is not Boolean true:
  1. Append "=" to output.
  2. Append the result of running Serializing a bare Item (Section 4.1.3.1) with `param_value` to output.
3. Return output.

#### 4.1.1.3. Serializing a Key

Given a key as `input_key`, return an ASCII string suitable for use in a HTTP field value.

1. Convert `input_key` into a sequence of ASCII characters; if conversion fails, fail serialization.
2. If `input_key` contains characters not in `lcalpha`, `DIGIT`, "\_", "-", ".", or "\*" fail serialization.
3. If the first character of `input_key` is not `lcalpha` or "\*", fail serialization.
4. Let output be an empty string.
5. Append `input_key` to output.
6. Return output.

#### 4.1.2. Serializing a Dictionary

Given an ordered Dictionary as `input_dictionary` (each member having a `member_name` and a tuple value of (`member_value`, `parameters`)), return an ASCII string suitable for use in a HTTP field value.

1. Let output be an empty string.
2. For each `member_name` with a value of (`member_value`, `parameters`) in `input_dictionary`:
  1. Append the result of running Serializing a Key (Section 4.1.1.3) with `member's member_name` to output.
  2. If `member_value` is Boolean true:

1. Append the result of running Serializing Parameters (Section 4.1.1.2) with parameters to output.
3. Otherwise:
  1. Append "=" to output.
  2. If member\_value is an array, append the result of running Serializing an Inner List (Section 4.1.1.1) with (member\_value, parameters) to output.
  3. Otherwise, append the result of running Serializing an Item (Section 4.1.3) with (member\_value, parameters) to output.
4. If more members remain in input\_dictionary:
  1. Append "," to output.
  2. Append a single SP to output.
3. Return output.

#### 4.1.3. Serializing an Item

Given an Item as bare\_item and Parameters as item\_parameters, return an ASCII string suitable for use in a HTTP field value.

1. Let output be an empty string.
2. Append the result of running Serializing a Bare Item Section 4.1.3.1 with bare\_item to output.
3. Append the result of running Serializing Parameters Section 4.1.1.2 with item\_parameters to output.
4. Return output.

##### 4.1.3.1. Serializing a Bare Item

Given an Item as input\_item, return an ASCII string suitable for use in a HTTP field value.

1. If input\_item is an Integer, return the result of running Serializing an Integer (Section 4.1.4) with input\_item.
2. If input\_item is a Decimal, return the result of running Serializing a Decimal (Section 4.1.5) with input\_item.

3. If `input_item` is a String, return the result of running Serializing a String (Section 4.1.6) with `input_item`.
4. If `input_item` is a Token, return the result of running Serializing a Token (Section 4.1.7) with `input_item`.
5. If `input_item` is a Boolean, return the result of running Serializing a Boolean (Section 4.1.9) with `input_item`.
6. If `input_item` is a Byte Sequence, return the result of running Serializing a Byte Sequence (Section 4.1.8) with `input_item`.
7. Otherwise, fail serialization.

#### 4.1.4. Serializing an Integer

Given an Integer as `input_integer`, return an ASCII string suitable for use in a HTTP field value.

1. If `input_integer` is not an integer in the range of -999,999,999,999,999 to 999,999,999,999,999 inclusive, fail serialization.
2. Let `output` be an empty string.
3. If `input_integer` is less than (but not equal to) 0, append "-" to `output`.
4. Append `input_integer`'s numeric value represented in base 10 using only decimal digits to `output`.
5. Return `output`.

#### 4.1.5. Serializing a Decimal

Given a decimal number as `input_decimal`, return an ASCII string suitable for use in a HTTP field value.

1. If `input_decimal` is not a decimal number, fail serialization.
2. If `input_decimal` has more than three significant digits to the right of the decimal point, round it to three decimal places, rounding the final digit to the nearest value, or to the even value if it is equidistant.
3. If `input_decimal` has more than 12 significant digits to the left of the decimal point after rounding, fail serialization.

4. Let output be an empty string.
5. If input\_decimal is less than (but not equal to) 0, append "-" to output.
6. Append input\_decimal's integer component represented in base 10 (using only decimal digits) to output; if it is zero, append "0".
7. Append "." to output.
8. If input\_decimal's fractional component is zero, append "0" to output.
9. Otherwise, append the significant digits of input\_decimal's fractional component represented in base 10 (using only decimal digits) to output.
10. Return output.

#### 4.1.6. Serializing a String

Given a String as input\_string, return an ASCII string suitable for use in a HTTP field value.

1. Convert input\_string into a sequence of ASCII characters; if conversion fails, fail serialization.
2. If input\_string contains characters in the range %x00-1f or %x7f (i.e., not in VCHAR or SP), fail serialization.
3. Let output be the string DQUOTE.
4. For each character char in input\_string:
  1. If char is "\" or DQUOTE:
    1. Append "\" to output.
  2. Append char to output.
5. Append DQUOTE to output.
6. Return output.

#### 4.1.7. Serializing a Token

Given a Token as `input_token`, return an ASCII string suitable for use in a HTTP field value.

1. Convert `input_token` into a sequence of ASCII characters; if conversion fails, fail serialization.
2. If the first character of `input_token` is not ALPHA or "\*", or the remaining portion contains a character not in `tchar`, ":" or "/", fail serialization.
3. Let `output` be an empty string.
4. Append `input_token` to `output`.
5. Return `output`.

#### 4.1.8. Serializing a Byte Sequence

Given a Byte Sequence as `input_bytes`, return an ASCII string suitable for use in a HTTP field value.

1. If `input_bytes` is not a sequence of bytes, fail serialization.
2. Let `output` be an empty string.
3. Append ":" to `output`.
4. Append the result of base64-encoding `input_bytes` as per [RFC4648], Section 4, taking account of the requirements below.
5. Append ":" to `output`.
6. Return `output`.

The encoded data is required to be padded with "=", as per [RFC4648], Section 3.2.

Likewise, encoded data SHOULD have pad bits set to zero, as per [RFC4648], Section 3.5, unless it is not possible to do so due to implementation constraints.

#### 4.1.9. Serializing a Boolean

Given a Boolean as `input_boolean`, return an ASCII string suitable for use in a HTTP field value.

1. If `input_boolean` is not a boolean, fail serialization.
2. Let `output` be an empty string.
3. Append "?" to `output`.
4. If `input_boolean` is true, append "1" to `output`.
5. If `input_boolean` is false, append "0" to `output`.
6. Return `output`.

#### 4.2. Parsing Structured Fields

When a receiving implementation parses HTTP fields that are known to be Structured Fields, it is important that care be taken, as there are a number of edge cases that can cause interoperability or even security problems. This section specifies the algorithm for doing so.

Given an array of bytes `input_bytes` that represents the chosen field's field-value (which is empty if that field is not present), and `field_type` (one of "dictionary", "list", or "item"), return the parsed header value.

1. Convert `input_bytes` into an ASCII string `input_string`; if conversion fails, fail parsing.
2. Discard any leading SP characters from `input_string`.
3. If `field_type` is "list", let `output` be the result of running Parsing a List (Section 4.2.1) with `input_string`.
4. If `field_type` is "dictionary", let `output` be the result of running Parsing a Dictionary (Section 4.2.2) with `input_string`.
5. If `field_type` is "item", let `output` be the result of running Parsing an Item (Section 4.2.3) with `input_string`.
6. Discard any leading SP characters from `input_string`.
7. If `input_string` is not empty, fail parsing.
8. Otherwise, return `output`.

When generating `input_bytes`, parsers MUST combine all field lines in the same section (header or trailer) that case-insensitively match the field name into one comma-separated field-value, as per



[RFC7230], Section 3.2.2; this assures that the entire field value is processed correctly.

For Lists and Dictionaries, this has the effect of correctly concatenating all of the field's lines, as long as individual members of the top-level data structure are not split across multiple header instances. The parsing algorithms for both types allow tab characters, since these might be used to combine field lines by some implementations.

Strings split across multiple field lines will have unpredictable results, because comma(s) and whitespace inserted upon combination will become part of the string output by the parser. Since concatenation might be done by an upstream intermediary, the results are not under the control of the serializer or the parser, even when they are both under the control of the same party.

Tokens, Integers, Decimals and Byte Sequences cannot be split across multiple field lines because the inserted commas will cause parsing to fail.

Parsers MAY fail when processing a field value spread across multiple field lines, when one of those lines does not parse as that field. For example, a parsing handling an Example-String field that's defined as a sf-string is allowed to fail when processing this field section:

```
Example-String: "foo
Example-String: bar"
```

If parsing fails - including when calling another algorithm - the entire field value MUST be ignored (i.e., treated as if the field were not present in the section). This is intentionally strict, to improve interoperability and safety, and specifications referencing this document are not allowed to loosen this requirement.

Note that this requirement does not apply to an implementation that is not parsing the field; for example, an intermediary is not required to strip a failing field from a message before forwarding it.

#### 4.2.1. Parsing a List

Given an ASCII string as `input_string`, return an array of (`item_or_inner_list`, `parameters`) tuples. `input_string` is modified to remove the parsed value.

1. Let `members` be an empty array.

2. While `input_string` is not empty:
  1. Append the result of running Parsing an Item or Inner List (Section 4.2.1.1) with `input_string` to `members`.
  2. Discard any leading OWS characters from `input_string`.
  3. If `input_string` is empty, return `members`.
  4. Consume the first character of `input_string`; if it is not `"`, fail parsing.
  5. Discard any leading OWS characters from `input_string`.
  6. If `input_string` is empty, there is a trailing comma; fail parsing.
3. No structured data has been found; return `members` (which is empty).

#### 4.2.1.1. Parsing an Item or Inner List

Given an ASCII string as `input_string`, return the tuple (`item_or_inner_list`, `parameters`), where `item_or_inner_list` can be either a single bare item, or an array of (`bare_item`, `parameters`) tuples. `input_string` is modified to remove the parsed value.

1. If the first character of `input_string` is `"`, return the result of running Parsing an Inner List (Section 4.2.1.2) with `input_string`.
2. Return the result of running Parsing an Item (Section 4.2.3) with `input_string`.

#### 4.2.1.2. Parsing an Inner List

Given an ASCII string as `input_string`, return the tuple (`inner_list`, `parameters`), where `inner_list` is an array of (`bare_item`, `parameters`) tuples. `input_string` is modified to remove the parsed value.

1. Consume the first character of `input_string`; if it is not `"`, fail parsing.
2. Let `inner_list` be an empty array.
3. While `input_string` is not empty:
  1. Discard any leading SP characters from `input_string`.

2. If the first character of `input_string` is `"")`:
    1. Consume the first character of `input_string`.
    2. Let `parameters` be the result of running Parsing Parameters (Section 4.2.3.2) with `input_string`.
    3. Return the tuple (`inner_list`, `parameters`).
  3. Let `item` be the result of running Parsing an Item (Section 4.2.3) with `input_string`.
  4. Append `item` to `inner_list`.
  5. If the first character of `input_string` is not SP or `"")`, fail parsing.
4. The end of the inner list was not found; fail parsing.

#### 4.2.2. Parsing a Dictionary

Given an ASCII string as `input_string`, return an ordered map whose values are (`item_or_inner_list`, `parameters`) tuples. `input_string` is modified to remove the parsed value.

1. Let `dictionary` be an empty, ordered map.
2. While `input_string` is not empty:
  1. Let `this_key` be the result of running Parsing a Key (Section 4.2.3.3) with `input_string`.
  2. If the first character of `input_string` is `"="`:
    1. Consume the first character of `input_string`.
    2. Let `member` be the result of running Parsing an Item or Inner List (Section 4.2.1.1) with `input_string`.
  3. Otherwise:
    1. Let `value` be Boolean true.
    2. Let `parameters` be the result of running Parsing Parameters Section 4.2.3.2 with `input_string`.
    3. Let `member` be the tuple (`value`, `parameters`).

4. Add name `this_key` with value member to dictionary. If dictionary already contains a name `this_key` (comparing character-for-character), overwrite its value.
  5. Discard any leading OWS characters from `input_string`.
  6. If `input_string` is empty, return dictionary.
  7. Consume the first character of `input_string`; if it is not `",`, fail parsing.
  8. Discard any leading OWS characters from `input_string`.
  9. If `input_string` is empty, there is a trailing comma; fail parsing.
3. No structured data has been found; return dictionary (which is empty).

Note that when duplicate Dictionary keys are encountered, this has the effect of ignoring all but the last instance.

#### 4.2.3. Parsing an Item

Given an ASCII string as `input_string`, return a (`bare_item`, `parameters`) tuple. `input_string` is modified to remove the parsed value.

1. Let `bare_item` be the result of running Parsing a Bare Item (Section 4.2.3.1) with `input_string`.
2. Let `parameters` be the result of running Parsing Parameters (Section 4.2.3.2) with `input_string`.
3. Return the tuple (`bare_item`, `parameters`).

##### 4.2.3.1. Parsing a Bare Item

Given an ASCII string as `input_string`, return a bare Item. `input_string` is modified to remove the parsed value.

1. If the first character of `input_string` is a `"-` or a DIGIT, return the result of running Parsing an Integer or Decimal (Section 4.2.4) with `input_string`.
2. If the first character of `input_string` is a DQUOTE, return the result of running Parsing a String (Section 4.2.5) with `input_string`.

3. If the first character of `input_string` is ":", return the result of running Parsing a Byte Sequence (Section 4.2.7) with `input_string`.
4. If the first character of `input_string` is "?", return the result of running Parsing a Boolean (Section 4.2.8) with `input_string`.
5. If the first character of `input_string` is an ALPHA or "\*", return the result of running Parsing a Token (Section 4.2.6) with `input_string`.
6. Otherwise, the item type is unrecognized; fail parsing.

#### 4.2.3.2. Parsing Parameters

Given an ASCII string as `input_string`, return an ordered map whose values are bare Items. `input_string` is modified to remove the parsed value.

1. Let `parameters` be an empty, ordered map.
2. While `input_string` is not empty:
  1. If the first character of `input_string` is not ";", exit the loop.
  2. Consume a ";" character from the beginning of `input_string`.
  3. Discard any leading SP characters from `input_string`.
  4. let `param_name` be the result of running Parsing a Key (Section 4.2.3.3) with `input_string`.
  5. Let `param_value` be Boolean true.
  6. If the first character of `input_string` is "=:
    1. Consume the "=" character at the beginning of `input_string`.
    2. Let `param_value` be the result of running Parsing a Bare Item (Section 4.2.3.1) with `input_string`.
  7. Append key `param_name` with value `param_value` to `parameters`. If `parameters` already contains a name `param_name` (comparing character-for-character), overwrite its value.
3. Return `parameters`.

Note that when duplicate Parameter keys are encountered, this has the effect of ignoring all but the last instance.

#### 4.2.3.3. Parsing a Key

Given an ASCII string as `input_string`, return a key. `input_string` is modified to remove the parsed value.

1. If the first character of `input_string` is not `lcalpha` or `"*"`, fail parsing.
2. Let `output_string` be an empty string.
3. While `input_string` is not empty:
  1. If the first character of `input_string` is not one of `lcalpha`, `DIGIT`, `"_"`, `"-"`, `"."`, or `"*"`, return `output_string`.
  2. Let `char` be the result of consuming the first character of `input_string`.
  3. Append `char` to `output_string`.
4. Return `output_string`.

#### 4.2.4. Parsing an Integer or Decimal

Given an ASCII string as `input_string`, return an Integer or Decimal. `input_string` is modified to remove the parsed value.

NOTE: This algorithm parses both Integers (Section 3.3.1) and Decimals (Section 3.3.2), and returns the corresponding structure.

1. Let `type` be `"integer"`.
2. Let `sign` be 1.
3. Let `input_number` be an empty string.
4. If the first character of `input_string` is `"-"`, consume it and set `sign` to -1.
5. If `input_string` is empty, there is an empty integer; fail parsing.
6. If the first character of `input_string` is not a `DIGIT`, fail parsing.

7. While `input_string` is not empty:
    1. Let `char` be the result of consuming the first character of `input_string`.
    2. If `char` is a DIGIT, append it to `input_number`.
    3. Else, if type is "integer" and `char` is ".":
      1. If `input_number` contains more than 12 characters, fail parsing.
      2. Otherwise, append `char` to `input_number` and set type to "decimal".
    4. Otherwise, prepend `char` to `input_string`, and exit the loop.
    5. If type is "integer" and `input_number` contains more than 15 characters, fail parsing.
    6. If type is "decimal" and `input_number` contains more than 16 characters, fail parsing.
  8. If type is "integer":
    1. Parse `input_number` as an integer and let `output_number` be the product of the result and sign.
    2. If `output_number` is outside the range -999,999,999,999,999 to 999,999,999,999,999 inclusive, fail parsing.
  9. Otherwise:
    1. If the final character of `input_number` is ".", fail parsing.
    2. If the number of characters after "." in `input_number` is greater than three, fail parsing.
    3. Parse `input_number` as a decimal number and let `output_number` be the product of the result and sign.
  10. Return `output_number`.
- 4.2.5. Parsing a String

Given an ASCII string as `input_string`, return an unquoted String.  
`input_string` is modified to remove the parsed value.

1. Let `output_string` be an empty string.
2. If the first character of `input_string` is not `DQUOTE`, fail parsing.
3. Discard the first character of `input_string`.
4. While `input_string` is not empty:
  1. Let `char` be the result of consuming the first character of `input_string`.
  2. If `char` is a backslash (`"\"`):
    1. If `input_string` is now empty, fail parsing.
    2. Let `next_char` be the result of consuming the first character of `input_string`.
    3. If `next_char` is not `DQUOTE` or `"\"`, fail parsing.
    4. Append `next_char` to `output_string`.
  3. Else, if `char` is `DQUOTE`, return `output_string`.
  4. Else, if `char` is in the range `%x00-1f` or `%x7f` (i.e., is not in `VCHAR` or `SP`), fail parsing.
  5. Else, append `char` to `output_string`.
5. Reached the end of `input_string` without finding a closing `DQUOTE`; fail parsing.

#### 4.2.6. Parsing a Token

Given an ASCII string as `input_string`, return a Token. `input_string` is modified to remove the parsed value.

1. If the first character of `input_string` is not ALPHA or `"*"`, fail parsing.
2. Let `output_string` be an empty string.
3. While `input_string` is not empty:
  1. If the first character of `input_string` is not in `tchar`, `":"` or `"/"`, return `output_string`.



2. Let char be the result of consuming the first character of input\_string.
3. Append char to output\_string.
4. Return output\_string.

#### 4.2.7. Parsing a Byte Sequence

Given an ASCII string as input\_string, return a Byte Sequence. input\_string is modified to remove the parsed value.

1. If the first character of input\_string is not ":", fail parsing.
2. Discard the first character of input\_string.
3. If there is not a ":" character before the end of input\_string, fail parsing.
4. Let b64\_content be the result of consuming content of input\_string up to but not including the first instance of the character ":".
5. Consume the ":" character at the beginning of input\_string.
6. If b64\_content contains a character not included in ALPHA, DIGIT, "+", "/" and "=", fail parsing.
7. Let binary\_content be the result of Base 64 Decoding [RFC4648] b64\_content, synthesizing padding if necessary (note the requirements about recipient behavior below).
8. Return binary\_content.

Because some implementations of base64 do not allow rejection of encoded data that is not properly "=" padded (see [RFC4648], Section 3.2), parsers SHOULD NOT fail when "=" padding is not present, unless they cannot be configured to do so.

Because some implementations of base64 do not allow rejection of encoded data that has non-zero pad bits (see [RFC4648], Section 3.5), parsers SHOULD NOT fail when non-zero pad bits are present, unless they cannot be configured to do so.

This specification does not relax the requirements in [RFC4648], Section 3.1 and 3.3; therefore, parsers MUST fail on characters outside the base64 alphabet, and on line feeds in encoded data.

#### 4.2.8. Parsing a Boolean

Given an ASCII string as `input_string`, return a Boolean. `input_string` is modified to remove the parsed value.

1. If the first character of `input_string` is not "?", fail parsing.
2. Discard the first character of `input_string`.
3. If the first character of `input_string` matches "1", discard the first character, and return true.
4. If the first character of `input_string` matches "0", discard the first character, and return false.
5. No value has matched; fail parsing.

#### 5. IANA Considerations

This document has no actions for IANA.

#### 6. Security Considerations

The size of most types defined by Structured Fields is not limited; as a result, extremely large fields could be an attack vector (e.g., for resource consumption). Most HTTP implementations limit the sizes of individual fields as well as the overall header or trailer section size to mitigate such attacks.

It is possible for parties with the ability to inject new HTTP fields to change the meaning of a Structured Field. In some circumstances, this will cause parsing to fail, but it is not possible to reliably fail in all such circumstances.

#### 7. References

##### 7.1. Normative References

- [RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <<https://www.rfc-editor.org/info/rfc20>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 7.2. Informative References

- [IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2019, DOI 10.1109/IEEESTD.2019.8766229, ISBN 978-1-5044-5924-2, July 2019, <<https://ieeexplore.ieee.org/document/8766229>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <<https://www.rfc-editor.org/info/rfc7541>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<http://www.rfc-editor.org/info/std63>>.

### 7.3. URIs

- [1] <https://lists.w3.org/Archives/Public/ietf-http-wg/>
- [2] <https://httpwg.github.io/>
- [3] <https://github.com/httpwg/http-extensions/labels/header-structure>
- [4] <https://github.com/httpwg/structured-field-tests>
- [5] <https://github.com/httpwg/wiki/wiki/Structured-Headers>
- [6] <https://github.com/httpwg/structured-field-tests>

## Appendix A. Frequently Asked Questions

### A.1. Why not JSON?

Earlier proposals for Structured Fields were based upon JSON [RFC8259]. However, constraining its use to make it suitable for HTTP header fields required senders and recipients to implement specific additional handling.

For example, JSON has specification issues around large numbers and objects with duplicate members. Although advice for avoiding these issues is available (e.g., [RFC7493]), it cannot be relied upon.

Likewise, JSON strings are by default Unicode strings, which have a number of potential interoperability issues (e.g., in comparison). Although implementers can be advised to avoid non-ASCII content where unnecessary, this is difficult to enforce.

Another example is JSON's ability to nest content to arbitrary depths. Since the resulting memory commitment might be unsuitable (e.g., in embedded and other limited server deployments), it's necessary to limit it in some fashion; however, existing JSON implementations have no such limits, and even if a limit is specified, it's likely that some field definition will find a need to violate it.

Because of JSON's broad adoption and implementation, it is difficult to impose such additional constraints across all implementations; some deployments would fail to enforce them, thereby harming

interoperability. In short, if it looks like JSON, people will be tempted to use a JSON parser / serializer on field values.

Since a major goal for Structured Fields is to improve interoperability and simplify implementation, these concerns led to a format that requires a dedicated parser and serializer.

Additionally, there were widely shared feelings that JSON doesn't "look right" in HTTP fields.

## Appendix B. Implementation Notes

A generic implementation of this specification should expose the top-level `serialize` (Section 4.1) and `parse` (Section 4.2) functions. They need not be functions; for example, it could be implemented as an object, with methods for each of the different top-level types.

For interoperability, it's important that generic implementations be complete and follow the algorithms closely; see Section 1.1. To aid this, a common test suite is being maintained by the community at <https://github.com/httpwg/structured-field-tests> [6].

Implementers should note that Dictionaries and Parameters are order-preserving maps. Some fields may not convey meaning in the ordering of these data types, but it should still be exposed so that applications which need to use it will have it available.

Likewise, implementations should note that it's important to preserve the distinction between Tokens and Strings. While most programming languages have native types that map to the other types well, it may be necessary to create a wrapper "token" object or use a parameter on functions to assure that these types remain separate.

The serialization algorithm is defined in a way that it is not strictly limited to the data types defined in Section 3 in every case. For example, Decimals are designed to take broader input and round to allowed values.

Implementations are allowed to limit the allowed size of different structures, subject to the minimums defined for each type. When a structure exceeds an implementation limit, that structure fails parsing or serialisation.

## Appendix C. Changes

\_\_RFC Editor: Please remove this section before publication.\_\_

## C.1. Since draft-ietf-httpbis-header-structure-18

- o Use "sf-" prefix for ABNF, not "sh-".
- o Fix indentation in Dictionary serialisation (#1164).
- o Add example for Token; tweak example field names (#1147).
- o Editorial improvements.
- o Note that exceeding implementation limits implies failure.
- o Talk about specifying order of Dictionary members and Parameters, not cardinality.
- o Allow (but don't require) parsers to fail when a single field line isn't valid.
- o Note that some aspects of Integers and Decimals are not necessarily preserved.
- o Allow Lists and Dictionaries to be delimited by OWS, rather than \*SP, to make parsing more robust.

## C.2. Since draft-ietf-httpbis-header-structure-17

- o Editorial improvements.

## C.3. Since draft-ietf-httpbis-header-structure-16

- o Editorial improvements.
- o Discussion on forwards compatibility.

## C.4. Since draft-ietf-httpbis-header-structure-15

- o Editorial improvements.
- o Use HTTP field terminology more consistently, in line with recent changes to HTTP-core.
- o String length requirements apply to decoded strings (#1051).
- o Correctly round decimals in serialisation (#1043).
- o Clarify input to serialisation algorithms (#1055).
- o Omitted True dictionary value can have parameters (#1083).

- o Keys can now start with '\*' (#1068).
- C.5. Since draft-ietf-httpbis-header-structure-14
- o Editorial improvements.
  - o Allow empty dictionary values (#992).
  - o Change value of omitted parameter value to True (#995).
  - o Explain more about splitting dictionaries and lists across header instances (#997).
  - o Disallow HTAB, replace OWS with spaces (#998).
  - o Change byte sequence delimiters from "\*" to ":" (#991).
  - o Allow tokens to start with "\*" (#991).
  - o Change Floats to fixed-precision Decimals (#982).
  - o Round the fractional component of decimal, rather than truncating it (#982).
  - o Handle duplicate dictionary and parameter keys by overwriting their values, rather than failing (#997).
  - o Allow "." in key (#1027).
  - o Check first character of key in serialisation (#1037).
  - o Talk about greasing headers (#1015).
- C.6. Since draft-ietf-httpbis-header-structure-13
- o Editorial improvements.
  - o Define "structured header name" and "structured header value" terms (#908).
  - o Corrected text about valid characters in strings (#931).
  - o Removed most instances of the word "textual", as it was redundant (#915).
  - o Allowed parameters on Items and Inner Lists (#907).
  - o Expand the range of characters in token (#961).

- o Disallow OWS before ";" delimiter in parameters (#961).
- C.7. Since draft-ietf-httpbis-header-structure-12
- o Editorial improvements.
  - o Reworked float serialisation (#896).
  - o Don't add a trailing space in inner-list (#904).
- C.8. Since draft-ietf-httpbis-header-structure-11
- o Allow \* in key (#844).
  - o Constrain floats to six digits of precision (#848).
  - o Allow dictionary members to have parameters (#842).
- C.9. Since draft-ietf-httpbis-header-structure-10
- o Update abstract (#799).
  - o Input and output are now arrays of bytes (#662).
  - o Implementations need to preserve difference between token and string (#790).
  - o Allow empty dictionaries and lists (#781).
  - o Change parameterized lists to have primary items (#797).
  - o Allow inner lists in both dictionaries and lists; removes lists of lists (#816).
  - o Subsume Parameterised Lists into Lists (#839).
- C.10. Since draft-ietf-httpbis-header-structure-09
- o Changed Boolean from T/F to 1/0 (#784).
  - o Parameters are now ordered maps (#765).
  - o Clamp integers to 15 digits (#737).



## C.11. Since draft-ietf-httpbis-header-structure-08

- o Disallow whitespace before items properly (#703).
- o Created "key" for use in dictionaries and parameters, rather than relying on identifier (#702). Identifiers have a separate minimum supported size.
- o Expanded the range of special characters allowed in identifier to include all of ALPHA, ".", ":", and "%" (#702).
- o Use "?" instead of "!" to indicate a Boolean (#719).
- o Added "Intentionally Strict Processing" (#684).
- o Gave better names for referring specs to use in Parameterised Lists (#720).
- o Added Lists of Lists (#721).
- o Rename Identifier to Token (#725).
- o Add implementation guidance (#727).

## C.12. Since draft-ietf-httpbis-header-structure-07

- o Make Dictionaries ordered mappings (#659).
- o Changed "binary content" to "byte sequence" to align with Infra specification (#671).
- o Changed "mapping" to "map" for #671.
- o Don't fail if byte sequences aren't "=" padded (#658).
- o Add Booleans (#683).
- o Allow identifiers in items again (#629).
- o Disallowed whitespace before items (#703).
- o Explain the consequences of splitting a string across multiple headers (#686).

- C.13. Since draft-ietf-httpbis-header-structure-06
- o Add a FAQ.
  - o Allow non-zero pad bits.
  - o Explicitly check for integers that violate constraints.
- C.14. Since draft-ietf-httpbis-header-structure-05
- o Reorganise specification to separate parsing out.
  - o Allow referencing specs to use ABNF.
  - o Define serialisation algorithms.
  - o Refine relationship between ABNF, parsing and serialisation algorithms.
- C.15. Since draft-ietf-httpbis-header-structure-04
- o Remove identifiers from item.
  - o Remove most limits on sizes.
  - o Refine number parsing.
- C.16. Since draft-ietf-httpbis-header-structure-03
- o Strengthen language around failure handling.
- C.17. Since draft-ietf-httpbis-header-structure-02
- o Split Numbers into Integers and Floats.
  - o Define number parsing.
  - o Tighten up binary parsing and give it an explicit end delimiter.
  - o Clarify that mappings are unordered.
  - o Allow zero-length strings.
  - o Improve string parsing algorithm.
  - o Improve limits in algorithms.
  - o Require parsers to combine header fields before processing.

- o Throw an error on trailing garbage.
- C.18. Since draft-ietf-httpbis-header-structure-01
- o Replaced with draft-nottingham-structured-headers.
- C.19. Since draft-ietf-httpbis-header-structure-00
- o Added signed 64bit integer type.
  - o Drop UTF8, and settle on BCP137 ::EmbeddedUnicodeChar for hl-unicode-string.
  - o Change hl\_blob delimiter to ":" since "'" is valid t\_char

#### Acknowledgements

Many thanks to Matthew Kerwin for his detailed feedback and careful consideration during the development of this specification.

Thanks also to Ian Clelland, Roy Fielding, Anne van Kesteren, Kazuho Oku, Evert Pot, Julian Reschke, Martin Thomson, Mike West, and Jeffrey Yasskin for their contributions.

#### Authors' Addresses

Mark Nottingham  
Fastly

Email: [mnot@mnot.net](mailto:mnot@mnot.net)  
URI: <https://www.mnot.net/>

Poul-Henning Kamp  
The Varnish Cache Project

Email: [phk@varnish-cache.org](mailto:phk@varnish-cache.org)

HTTP  
Internet-Draft  
Intended status: Standards Track  
Expires: November 15, 2020

M. Bishop  
Akamai  
N. Sullivan  
Cloudflare  
M. Thomson  
Mozilla  
May 14, 2020

Secondary Certificate Authentication in HTTP/2  
draft-ietf-httpbis-http2-secondary-certs-06

Abstract

A use of TLS Exported Authenticators is described which enables HTTP/2 clients and servers to offer additional certificate-based credentials after the connection is established. The means by which these credentials are used with requests is defined.

Note to Readers

Discussion of this draft takes place on the HTTP working group mailing list ([ietf-http-wg@w3.org](mailto:ietf-http-wg@w3.org)), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/> [1].

Working Group information can be found at <http://httpwg.github.io/> [2]; source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions/labels/secondary-certs> [3].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 15, 2020.

## Copyright Notice

Copyright (c) 2020 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 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 . . . . .	3
1.1. Server Certificate Authentication . . . . .	4
1.2. Client Certificate Authentication . . . . .	4
1.2.1. HTTP/1.1 Using TLS 1.2 and Earlier . . . . .	5
1.2.2. HTTP/1.1 Using TLS 1.3 . . . . .	6
1.2.3. HTTP/2 . . . . .	6
1.3. HTTP-Layer Certificate Authentication . . . . .	7
1.4. Terminology . . . . .	8
2. Discovering Additional Certificates at the HTTP/2 Layer . . .	8
2.1. Indicating Support for HTTP-Layer Certificate Authentication . . . . .	9
2.2. Making Certificates or Requests Available . . . . .	10
2.3. Requiring Certificate Authentication . . . . .	11
2.3.1. Requiring Additional Server Certificates . . . . .	11
2.3.2. Requiring Additional Client Certificates . . . . .	12
3. Certificates Frames for HTTP/2 . . . . .	13
3.1. The CERTIFICATE_NEEDED Frame . . . . .	13
3.2. The USE_CERTIFICATE Frame . . . . .	15
3.3. The CERTIFICATE_REQUEST Frame . . . . .	16
3.3.1. Exported Authenticator Request Characteristics . . .	17
3.4. The CERTIFICATE Frame . . . . .	17
3.4.1. Exported Authenticator Characteristics . . . . .	19
4. Indicating Failures During HTTP-Layer Certificate Authentication . . . . .	19
4.1. Misbehavior . . . . .	20
4.2. Invalid Certificates . . . . .	20
5. Required Domain Certificate Extension . . . . .	20
6. Security Considerations . . . . .	21
6.1. Impersonation . . . . .	21
6.2. Fingerprinting . . . . .	22
6.3. Denial of Service . . . . .	22

6.4.	Persistence of Service . . . . .	22
6.5.	Confusion About State . . . . .	23
7.	IANA Considerations . . . . .	23
7.1.	HTTP/2 Settings . . . . .	23
7.2.	New HTTP/2 Frames . . . . .	24
7.3.	New HTTP/2 Error Codes . . . . .	24
8.	References . . . . .	24
8.1.	Normative References . . . . .	24
8.2.	Informative References . . . . .	25
8.3.	URIs . . . . .	26
Appendix A.	Change Log . . . . .	26
A.1.	Since draft-ietf-httpbis-http2-secondary-certs-04: . . .	26
A.2.	Since draft-ietf-httpbis-http2-secondary-certs-03: . . .	26
A.3.	Since draft-ietf-httpbis-http2-secondary-certs-02: . . .	26
A.4.	Since draft-ietf-httpbis-http2-secondary-certs-01: . . .	26
A.5.	Since draft-ietf-httpbis-http2-secondary-certs-00: . . .	27
A.6.	Since draft-bishop-httpbis-http2-additional-certs-05: . .	27
	Acknowledgements . . . . .	27
	Authors' Addresses . . . . .	27

## 1. Introduction

HTTP clients need to know that the content they receive on a connection comes from the origin that they intended to retrieve it from. The traditional form of server authentication in HTTP has been in the form of a single X.509 certificate provided during the TLS ([RFC5246], [RFC8446]) handshake.

Many existing HTTP [RFC7230] servers also have authentication requirements for the resources they serve. Of the bountiful authentication options available for authenticating HTTP requests, client certificates present a unique challenge for resource-specific authentication requirements because of the interaction with the underlying TLS layer.

TLS 1.2 [RFC5246] supports one server and one client certificate on a connection. These certificates may contain multiple identities, but only one certificate may be provided.

Many HTTP servers host content from several origins. HTTP/2 permits clients to reuse an existing HTTP connection to a server provided that the secondary origin is also in the certificate provided during the TLS handshake. In many cases, servers choose to maintain separate certificates for different origins but still desire the benefits of a shared HTTP connection.

### 1.1. Server Certificate Authentication

Section 9.1.1 of [RFC7540] describes how connections may be used to make requests from multiple origins as long as the server is authoritative for both. A server is considered authoritative for an origin if DNS resolves the origin to the IP address of the server and (for TLS) if the certificate presented by the server contains the origin in the Subject Alternative Names field.

[RFC7838] enables a step of abstraction from the DNS resolution. If both hosts have provided an Alternative Service at hostnames which resolve to the IP address of the server, they are considered authoritative just as if DNS resolved the origin itself to that address. However, the server's one TLS certificate is still required to contain the name of each origin in question.

[RFC8336] relaxes the requirement to perform the DNS lookup if already connected to a server with an appropriate certificate which claims support for a particular origin.

Servers which host many origins often would prefer to have separate certificates for some sets of origins. This may be for ease of certificate management (the ability to separately revoke or renew them), due to different sources of certificates (a CDN acting on behalf of multiple origins), or other factors which might drive this administrative decision. Clients connecting to such origins cannot currently reuse connections, even if both client and server would prefer to do so.

Because the TLS SNI extension is exchanged in the clear, clients might also prefer to retrieve certificates inside the encrypted context. When this information is sensitive, it might be advantageous to request a general-purpose certificate or anonymous ciphersuite at the TLS layer, while acquiring the "real" certificate in HTTP after the connection is established.

### 1.2. Client Certificate Authentication

For servers that wish to use client certificates to authenticate users, they might request client authentication during or immediately after the TLS handshake. However, if not all users or resources need certificate-based authentication, a request for a certificate has the unfortunate consequence of triggering the client to seek a certificate, possibly requiring user interaction, network traffic, or other time-consuming activities. During this time, the connection is stalled in many implementations. Such a request can result in a poor experience, particularly when sent to a client that does not expect the request.

The TLS 1.3 CertificateRequest can be used by servers to give clients hints about which certificate to offer. Servers that rely on certificate-based authentication might request different certificates for different resources. Such a server cannot use contextual information about the resource to construct an appropriate TLS CertificateRequest message during the initial handshake.

Consequently, client certificates are requested at connection establishment time only in cases where all clients are expected or required to have a single certificate that is used for all resources. Many other uses for client certificates are reactive, that is, certificates are requested in response to the client making a request.

#### 1.2.1. HTTP/1.1 Using TLS 1.2 and Earlier

In HTTP/1.1, a server that relies on client authentication for a subset of users or resources does not request a certificate when the connection is established. Instead, it only requests a client certificate when a request is made to a resource that requires a certificate. TLS 1.2 [RFC5246] accommodates this by permitting the server to request a new TLS handshake, in which the server will request the client's certificate.

Figure 1 shows the server initiating a TLS-layer renegotiation in response to receiving an HTTP/1.1 request to a protected resource.

Client	Server
-- (HTTP) GET /protected ----->	*1
<----- (TLS) HelloRequest --	*2
-- (TLS) ClientHello ----->	
<----- (TLS) ServerHello, ... --	
<----- (TLS) CertificateRequest --	*3
-- (TLS) ..., Certificate ----->	*4
-- (TLS) Finished ----->	
<----- (TLS) Finished --	
<----- (HTTP) 200 OK --	*5

Figure 1: HTTP/1.1 reactive certificate authentication with TLS 1.2

In this example, the server receives a request for a protected resource (at \*1 on Figure 1). Upon performing an authorization check, the server determines that the request requires authentication using a client certificate and that no such certificate has been provided.



The server initiates TLS renegotiation by sending a TLS HelloRequest (at \*2). The client then initiates a TLS handshake. Note that some TLS messages are elided from the figure for the sake of brevity.

The critical messages for this example are the server requesting a certificate with a TLS CertificateRequest (\*3); this request might use information about the request or resource. The client then provides a certificate and proof of possession of the private key in Certificate and CertificateVerify messages (\*4).

When the handshake completes, the server performs any authorization checks a second time. With the client certificate available, it then authorizes the request and provides a response (\*5).

### 1.2.2. HTTP/1.1 Using TLS 1.3

TLS 1.3 [RFC8446] introduces a new client authentication mechanism that allows for clients to authenticate after the handshake has been completed. For the purposes of authenticating an HTTP request, this is functionally equivalent to renegotiation. Figure 2 shows the simpler exchange this enables.

Client	Server
-- (HTTP) GET /protected	----->
<-----	(TLS) CertificateRequest --
-- (TLS) Certificate, CertificateVerify,	
Finished	----->
<-----	(HTTP) 200 OK --

Figure 2: HTTP/1.1 reactive certificate authentication with TLS 1.3

TLS 1.3 does not support renegotiation, instead supporting direct client authentication. In contrast to the TLS 1.2 example, in TLS 1.3, a server can simply request a certificate.

### 1.2.3. HTTP/2

An important part of the HTTP/1.1 exchange is that the client is able to easily identify the request that caused the TLS renegotiation. The client is able to assume that the next unanswered request on the connection is responsible. The HTTP stack in the client is then able to direct the certificate request to the application or component that initiated that request. This ensures that the application has the right contextual information for processing the request.

In HTTP/2, a client can have multiple outstanding requests. Without some sort of correlation information, a client is unable to identify which request caused the server to request a certificate.

Thus, the minimum necessary mechanism to support reactive certificate authentication in HTTP/2 is an identifier that can be used to correlate an HTTP request with a request for a certificate. Since streams are used for individual requests, correlation with a stream is sufficient.

[RFC7540] prohibits renegotiation after any application data has been sent. This completely blocks reactive certificate authentication in HTTP/2 using TLS 1.2. If this restriction were relaxed by an extension or update to HTTP/2, such an identifier could be added to TLS 1.2 by means of an extension to TLS. Unfortunately, many TLS 1.2 implementations do not permit application data to continue during a renegotiation. This is problematic for a multiplexed protocol like HTTP/2.

### 1.3. HTTP-Layer Certificate Authentication

This draft defines HTTP/2 frames to carry the relevant certificate messages, enabling certificate-based authentication of both clients and servers independent of TLS version. This mechanism can be implemented at the HTTP layer without breaking the existing interface between HTTP and applications above it.

This could be done in a naive manner by replicating the TLS messages as HTTP/2 frames on each stream. However, this would create needless redundancy between streams and require frequent expensive signing operations. Instead, TLS Exported Authenticators [I-D.ietf-tls-exported-authenticator] are exchanged on stream zero and other frames incorporate them to particular requests by reference as needed.

TLS Exported Authenticators are structured messages that can be exported by either party of a TLS connection and validated by the other party. Given an established TLS connection, a request can be constructed which describes the desired certificate and an authenticator message can be constructed proving possession of a certificate and a corresponding private key. Both requests and authenticators can be generated by either the client or the server. Exported Authenticators use the message structures from Sections 4.3.2 and 4.4 of [RFC8446], but different parameters.

Each Authenticator is computed using a Handshake Context and Finished MAC Key derived from the TLS session. The Handshake Context is identical for both parties of the TLS connection, while the Finished MAC Key is dependent on whether the Authenticator is created by the client or the server.

Successfully verified Authenticators result in certificate chains, with verified possession of the corresponding private key, which can be supplied into a collection of available certificates. Likewise, descriptions of desired certificates can be supplied into these collections.

Section 2 describes how the feature is employed, defining means to detect support in peers (Section 2.1), make certificates and requests available (Section 2.2), and indicate when streams are blocked waiting on an appropriate certificate (Section 2.3). Section 3 defines the required frame types, which parallel the TLS 1.3 message exchange. Finally, Section 4 defines new error types which can be used to notify peers when the exchange has not been successful.

#### 1.4. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

#### 2. Discovering Additional Certificates at the HTTP/2 Layer

A certificate chain with proof of possession of the private key corresponding to the end-entity certificate is sent as a sequence of "CERTIFICATE" frames (see Section 3.4) on stream zero. Once the holder of a certificate has sent the chain and proof, this certificate chain is cached by the recipient and available for future use. Clients can proactively indicate the certificate they intend to use on each request using an unsolicited "USE\_CERTIFICATE" frame, if desired. The previously-supplied certificates are available for reference without having to resend them.

Otherwise, the server uses a "CERTIFICATE\_REQUEST" frame to describe a class of certificates on stream zero, then uses "CERTIFICATE\_NEEDED" frames to associate these with individual requests. The client responds with a "USE\_CERTIFICATE" frame indicating the certificate which should be used to satisfy the request.

Data sent by each peer is correlated by the ID given in each frame. This ID is unrelated to values used by the other peer, even if each uses the same ID in certain cases. "USE\_CERTIFICATE" frames indicate whether they are sent proactively or are in response to a "CERTIFICATE\_NEEDED" frame.

## 2.1. Indicating Support for HTTP-Layer Certificate Authentication

Clients and servers that will accept requests for HTTP-layer certificate authentication indicate this using the HTTP/2 "SETTINGS\_HTTP\_CLIENT\_CERT\_AUTH" (0xSETTING-TBD1) and "SETTINGS\_HTTP\_SERVER\_CERT\_AUTH" (0xSETTING-TBD2) settings.

The initial value for both settings is 0, indicating that the peer does not support HTTP-layer certificate authentication. If a peer does support HTTP-layer certificate authentication, one or both of the values is non-zero. "SETTINGS\_HTTP\_CLIENT\_CERT\_AUTH" indicates that servers can use certificates for client authentication, while "SETTINGS\_HTTP\_SERVER\_CERT\_AUTH" indicates that servers are able to offer additional certificates to demonstrate control over other origin hostnames.

In order to ensure that the TLS connection is direct to the server, rather than via a TLS-terminating proxy, each side will separately compute and confirm the value of these settings. The setting values are derived from a TLS exporter (see Section 7.5 of [RFC8446] and [RFC5705] for more details on exporters). Clients MUST NOT use an early exporter during their 0-RTT flight, but MUST send an updated SETTINGS frame using a regular exporter after the TLS handshake completes.

The exporter is constructed with the following input:

- o Label:
  - \* "EXPORTER HTTP CERTIFICATE client" for clients
  - \* "EXPORTER HTTP CERTIFICATE server" for servers
- o Context: Empty
- o Length: Eight bytes

The value of the exporter is split into two four-byte values. The first four bytes are used for the "SETTINGS\_HTTP\_CLIENT\_CERT\_AUTH" value, while the following four bytes are used for the "SETTINGS\_HTTP\_SERVER\_CERT\_AUTH" value.

Each is converted to a setting value as:

Exporter fragment | 0x80000000

That is, the most significant bit will always be set, regardless of the value of the exporter. Each endpoint will compute the expected

values from their peer. If the setting is not received, or if the value received is not the expected value, the frames defined in this document SHOULD NOT be sent in the indicated direction.

## 2.2. Making Certificates or Requests Available

When both peers have advertised support for HTTP-layer certificates in a given direction as in Section 2.1, the indicated endpoint can supply additional certificates into the connection at any time. That is, if both endpoints have sent "SETTINGS\_HTTP\_SERVER\_CERT\_AUTH" and validated the value received from the peer, the server may send certificates. If both endpoints have sent "SETTINGS\_HTTP\_CLIENT\_CERT\_AUTH" and validated the value received from the peer, the client may send certificates.

Implementations which predict a certificate will be required could supply the certificate before being asked. These certificates are available for reference by future "USE\_CERTIFICATE" frames.

Certificates supplied by servers can be considered by clients without further action by the server. A server SHOULD NOT send certificates which do not cover origins which it is prepared to service on the current connection, but MAY use the ORIGIN frame [RFC8336] to indicate that not all covered origins will be served.

Certificates supplied by clients MUST NOT be considered by servers when processing a request unless the client explicitly authorizes their use. Clients MAY send "USE\_CERTIFICATE" frame with the "UNSOLICITED" flag set to indicate that an available certificate should be considered on a new request.

Client	Server
<----- (stream 0) CERTIFICATE --	
...	
-- (stream N) GET /from-new-origin ----->	
<----- (stream N) 200 OK --	

Figure 3: Proactive server authentication

Client	Server
-- (stream 0) CERTIFICATE ----->	
-- (stream 0) USE_CERTIFICATE (S=1) ----->	
-- (stream 0) USE_CERTIFICATE (S=3) ----->	
-- (streams 1,3) GET /protected ----->	
<----- (streams 1,3) 200 OK --	

Figure 4: Proactive client authentication

Likewise, a party can supply a "CERTIFICATE\_REQUEST" that outlines parameters of a certificate they might request in the future. Upon receipt of a "CERTIFICATE\_REQUEST", endpoints SHOULD provide a corresponding certificate in anticipation of a request shortly being blocked. Clients MAY wait for a "CERTIFICATE\_NEEDED" frame to assist in associating the certificate request with a particular HTTP transaction.

### 2.3. Requiring Certificate Authentication

#### 2.3.1. Requiring Additional Server Certificates

As defined in [RFC7540], when a client finds that an https:// origin (or Alternative Service [RFC7838]) to which it needs to make a request has the same IP address as a server to which it is already connected, it MAY check whether the TLS certificate provided contains the new origin as well, and if so, reuse the connection.

If the TLS certificate does not contain the new origin, but the server has claimed support for that origin (with an ORIGIN frame, see [RFC8336]) and advertised support for HTTP-layer server certificates (see Section 2.1), the client MAY send a "CERTIFICATE\_REQUEST" frame describing the desired origin. The client then sends a "CERTIFICATE\_NEEDED" frame for stream zero referencing the request, indicating that the connection cannot be used for that origin until the certificate is provided.

If the server does not have the desired certificate, it MUST send an Empty Authenticator, as described in Section 5 of [I-D.ietf-tls-exported-authenticator], in a "CERTIFICATE" frame in response to the request, followed by a "USE\_CERTIFICATE" frame for stream zero which references the Empty Authenticator.

If a server has not advertised support for HTTP-layer certificates, fails to provide a requested certificate, or provides a certificate which is unacceptable to the client, the client MUST NOT send any requests for resources in that origin on the current connection. The

client MAY open a new connection in an effort to reach an authoritative server.

Client	Server
	<----- (stream 0) ORIGIN --
	-- (stream 0) CERTIFICATE_REQUEST ----->
	-- (stream 0) CERTIFICATE_NEEDED (S=0) ----->
	<----- (stream 0) CERTIFICATE --
	<----- (stream 0) USE_CERTIFICATE (S=0) --
	-- (stream N) GET /from-new-origin ----->
	<----- (stream N) 200 OK --

Figure 5: Client-requested certificate

If a client receives a "PUSH\_PROMISE" referencing an origin for which it has not yet received the server's certificate, this is a stream error on the push stream; see section 8.2 of [RFC7540]. To avoid this, servers MUST supply the associated certificates before pushing resources from a different origin.

### 2.3.2. Requiring Additional Client Certificates

Likewise, the server sends a "CERTIFICATE\_NEEDED" frame for each stream where certificate authentication is required. The client answers with a "USE\_CERTIFICATE" frame indicating the certificate to use on that stream. If the request parameters or the responding certificate are not already available, they will need to be sent as described in Section 2.2 as part of this exchange.

Client	Server
	<----- (stream 0) CERTIFICATE_REQUEST --
	...
	-- (stream N) GET /protected ----->
	<----- (stream 0) CERTIFICATE_NEEDED (S=N) --
	-- (stream 0) CERTIFICATE ----->
	-- (stream 0) USE_CERTIFICATE (S=N) ----->
	<----- (stream N) 200 OK --

Figure 6: Reactive certificate authentication

If the client does not have the desired certificate, it instead sends an Empty Authenticator, as described in Section 5 of [I-D.ietf-tls-exported-authenticator], in a "CERTIFICATE" frame in response to the request, followed by a "USE\_CERTIFICATE" frame which references the Empty Authenticator.

If the client has not advertised support for HTTP-layer certificates, fails to provide a requested certificate, or provides a certificate

the server is unable to verify, the server processes the request based solely on the certificate provided during the TLS handshake, if any. This might result in an error response via HTTP, such as a status code 403 (Not Authorized).

### 3. Certificates Frames for HTTP/2

The "CERTIFICATE\_REQUEST" and "CERTIFICATE\_NEEDED" frames are correlated by their "Request-ID" field. Subsequent "CERTIFICATE\_NEEDED" frames with the same "Request-ID" value MAY be sent for other streams where the sender is expecting a certificate with the same parameters.

The "CERTIFICATE", and "USE\_CERTIFICATE" frames are correlated by their "Cert-ID" field. Subsequent "USE\_CERTIFICATE" frames with the same "Cert-ID" MAY be sent in response to other "CERTIFICATE\_NEEDED" frames and refer to the same certificate.

"CERTIFICATE\_NEEDED" and "USE\_CERTIFICATE" frames are correlated by the Stream ID they reference. Unsolicited "USE\_CERTIFICATE" frames are not responses to "CERTIFICATE\_NEEDED" frames; otherwise, each "USE\_CERTIFICATE" frame for a stream is considered to respond to a "CERTIFICATE\_NEEDED" frame for the same stream in sequence.

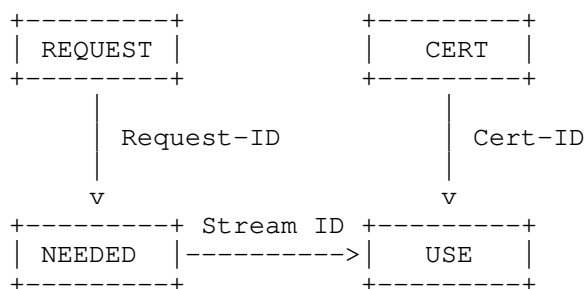


Figure 7: Frame correlation

"Request-ID" and "Cert-ID" are independent and sender-local. The use of the same value by the other peer or in the other context does not imply any correlation between these frames. These values MUST be unique per sender for each space over the lifetime of the connection.

#### 3.1. The CERTIFICATE\_NEEDED Frame

The "CERTIFICATE\_NEEDED" frame (0xFRAME-TBD1) is sent on stream zero to indicate that the HTTP request on the indicated stream is blocked pending certificate authentication. The frame includes stream ID and a request identifier which can be used to correlate the stream with a



previous "CERTIFICATE\_REQUEST" frame sent on stream zero. The "CERTIFICATE\_REQUEST" describes the certificate the sender requires to make progress on the stream in question.

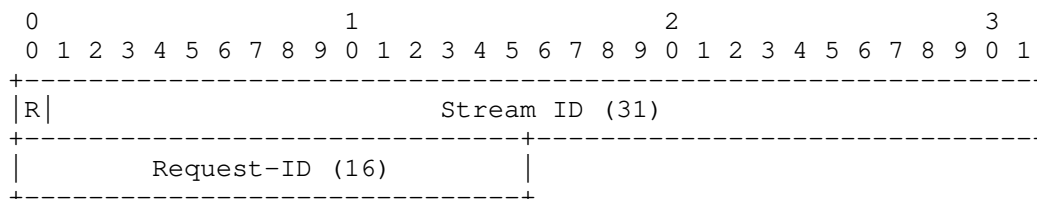


Figure 8: CERTIFICATE\_NEEDED frame payload

The "CERTIFICATE\_NEEDED" frame contains 6 octets. The first four octets indicate the Stream ID of the affected stream. The following two octets are the authentication request identifier, "Request-ID". A peer that receives a "CERTIFICATE\_NEEDED" of any other length MUST treat this as a stream error of type "PROTOCOL\_ERROR". Frames with identical request identifiers refer to the same "CERTIFICATE\_REQUEST".

A server MAY send multiple "CERTIFICATE\_NEEDED" frames for the same stream. If a server requires that a client provide multiple certificates before authorizing a single request, each required certificate MUST be indicated with a separate "CERTIFICATE\_NEEDED" frame, each of which MUST have a different request identifier (referencing different "CERTIFICATE\_REQUEST" frames describing each required certificate). To reduce the risk of client confusion, servers SHOULD NOT have multiple outstanding "CERTIFICATE\_NEEDED" frames for the same stream at any given time.

Clients MUST only send multiple "CERTIFICATE\_NEEDED" frames for stream zero. Multiple "CERTIFICATE\_NEEDED" frames on any other stream MUST be considered a stream error of type "PROTOCOL\_ERROR".

The "CERTIFICATE\_NEEDED" frame MUST NOT be sent to a client which has not advertised the "SETTINGS\_HTTP\_CLIENT\_CERT\_AUTH", or to a server which has not advertised the "SETTINGS\_HTTP\_SERVER\_CERT\_AUTH" setting. An endpoint which receives a "CERTIFICATE\_NEEDED" frame but did not advertise support MAY treat this as a connection error of type "CERTIFICATE\_WITHOUT\_CONSENT".

The "CERTIFICATE\_NEEDED" frame MUST NOT reference a stream in the "half-closed (local)" or "closed" states [RFC7540]. A client that receives a "CERTIFICATE\_NEEDED" frame for a stream which is not in a valid state SHOULD treat this as a stream error of type "PROTOCOL\_ERROR".

### 3.2. The USE\_CERTIFICATE Frame

The "USE\_CERTIFICATE" frame (0xFRAME-TBD4) is sent on stream zero to indicate which certificate is being used on a particular request stream.

The "USE\_CERTIFICATE" frame defines a single flag:

**UNSOLICITED (0x01):** Indicates that no "CERTIFICATE\_NEEDED" frame has yet been received for this stream.

The payload of the "USE\_CERTIFICATE" frame is as follows:

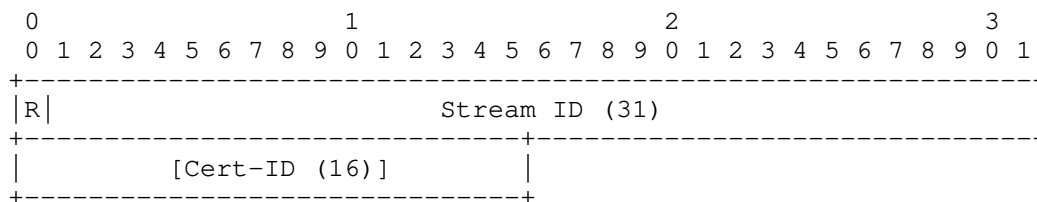


Figure 9: USE\_CERTIFICATE frame payload

The first four octets indicate the Stream ID of the affected stream. The following two octets, if present, contain the two-octet "Cert-ID" of the certificate the sender wishes to use. This MUST be the ID of a certificate for which proof of possession has been presented in a "CERTIFICATE" frame. Recipients of a "USE\_CERTIFICATE" frame of any other length MUST treat this as a stream error of type "PROTOCOL\_ERROR". Frames with identical certificate identifiers refer to the same certificate chain.

A "USE\_CERTIFICATE" frame which omits the Cert-ID refers to the certificate provided at the TLS layer, if any. If no certificate was provided at the TLS layer, the stream should be processed with no authentication, likely returning an authentication-related error at the HTTP level (e.g. 403) for servers or routing the request to a new connection for clients.

The "UNSOLICITED" flag MAY be set by clients on the first "USE\_CERTIFICATE" frame referring to a given stream. This permits a client to proactively indicate which certificate should be used when processing a new request. When such an unsolicited indication refers to a request that has not yet been received, servers SHOULD cache the indication briefly in anticipation of the request.

Receipt of more than one unsolicited "USE\_CERTIFICATE" frame or an unsolicited "USE\_CERTIFICATE" frame which is not the first in

reference to a given stream MUST be treated as a stream error of type "CERTIFICATE\_OVERUSED".

Each "USE\_CERTIFICATE" frame which is not marked as unsolicited is considered to respond in order to the "CERTIFICATE\_NEEDED" frames for the same stream. If a "USE\_CERTIFICATE" frame is received for which a "CERTIFICATE\_NEEDED" frame has not been sent, this MUST be treated as a stream error of type "CERTIFICATE\_OVERUSED".

Receipt of a "USE\_CERTIFICATE" frame with an unknown "Cert-ID" MUST result in a stream error of type "PROTOCOL\_ERROR".

The referenced certificate chain needs to conform to the requirements expressed in the "CERTIFICATE\_REQUEST" to the best of the sender's ability, or the recipient is likely to reject it as unsuitable despite properly validating the authenticator. If the recipient considers the certificate unsuitable, it MAY at its discretion either return an error at the HTTP semantic layer, or respond with a stream error [RFC7540] on any stream where the certificate is used. Section 4 defines certificate-related error codes which might be applicable.

### 3.3. The CERTIFICATE\_REQUEST Frame

The "CERTIFICATE\_REQUEST" frame (id=0xFRAME-TBD2) provides an exported authenticator request message from the TLS layer that specifies a desired certificate. This describes the certificate the sender wishes to have presented.

The "CERTIFICATE\_REQUEST" frame SHOULD NOT be sent to a client which has not advertised the "SETTINGS\_HTTP\_CLIENT\_CERT\_AUTH", or to a server which has not advertised the "SETTINGS\_HTTP\_SERVER\_CERT\_AUTH" setting.

The "CERTIFICATE\_REQUEST" frame MUST be sent on stream zero. A "CERTIFICATE\_REQUEST" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL\_ERROR".

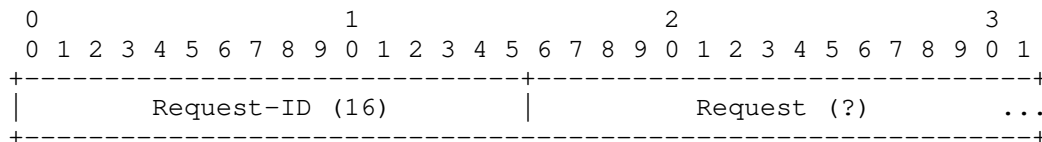


Figure 10: CERTIFICATE\_REQUEST frame payload

The frame contains the following fields:

**Request-ID:** "Request-ID" is a 16-bit opaque identifier used to correlate subsequent certificate-related frames with this request. The identifier **MUST** be unique in the session for the sender.

**Request:** An exported authenticator request, generated using the "request" API described in [I-D.ietf-tls-exported-authenticator]. See Section 3.4.1 for more details on the input to this API.

### 3.3.1. Exported Authenticator Request Characteristics

The Exported Authenticator "request" API defined in [I-D.ietf-tls-exported-authenticator] takes as input a set of desired certificate characteristics and a "certificate\_request\_context", which needs to be unpredictable. When generating exported authenticators for use with this extension, the "certificate\_request\_context" **MUST** contain both the two-octet Request-ID as well as at least 96 bits of additional entropy.

Upon receipt of a "CERTIFICATE\_REQUEST" frame, the recipient **MUST** verify that the first two octets of the authenticator's "certificate\_request\_context" matches the Request-ID presented in the frame.

The TLS library on the authenticating peer will provide mechanisms to select an appropriate certificate to respond to the transported request. TLS libraries on servers **MUST** be able to recognize the "server\_name" extension ([RFC6066]) at a minimum. Clients **MUST** always specify the desired origin using this extension, though other extensions **MAY** also be included.

### 3.4. The CERTIFICATE Frame

The "CERTIFICATE" frame (id=0xFRAME-TBD3) provides an exported authenticator message from the TLS layer that provides a chain of certificates, associated extensions and proves possession of the private key corresponding to the end-entity certificate.

The "CERTIFICATE" frame defines two flags:

**TO\_BE\_CONTINUED** (0x01): Indicates that the exported authenticator spans more than one frame.

**UNSOLICITED** (0x02): Indicates that the exported authenticator does not contain a Request-ID.

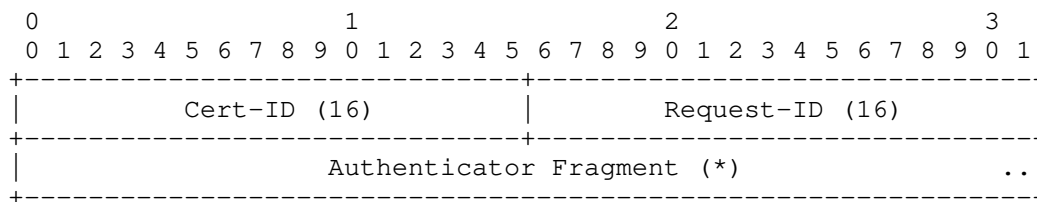


Figure 11: CERTIFICATE frame payload

The frame contains the following fields:

**Cert-ID:** "Cert-ID" is a 16-bit opaque identifier used to correlate other certificate- related frames with this exported authenticator fragment.

**Request-ID:** "Request-ID" is an optional 16-bit opaque identifier used to correlate this exported authenticator with the request which triggered it, if any. This field is present only if the "UNSOLICITED" flag is not set.

**Authenticator Fragment:** A portion of the opaque data returned from the TLS connection exported authenticator "authenticate" API. See Section 3.4.1 for more details on the input to this API.

An exported authenticator is transported in zero or more "CERTIFICATE" frames with the "TO\_BE\_CONTINUED" flag set, followed by one "CERTIFICATE" frame with the "TO\_BE\_CONTINUED" flag unset. Each of these frames contains the same "Cert-ID" field, permitting them to be associated with each other. Receipt of any "CERTIFICATE" frame with the same "Cert-ID" following the receipt of a "CERTIFICATE" frame with "TO\_BE\_CONTINUED" unset MUST be treated as a connection error of type "PROTOCOL\_ERROR".

If the "UNSOLICITED" flag is not set, the "CERTIFICATE" frame also contains a Request-ID indicating the certificate request which caused this exported authenticator to be generated. The value of this flag and the contents of the Request-ID field MUST NOT differ between frames with the same Cert-ID.

Upon receiving a complete series of "CERTIFICATE" frames, the receiver may validate the Exported Authenticator value by using the exported authenticator API. This returns either an error indicating that the message was invalid, or the certificate chain and extensions used to create the message.

The "CERTIFICATE" frame MUST be sent on stream zero. A "CERTIFICATE" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL\_ERROR".

#### 3.4.1. Exported Authenticator Characteristics

The Exported Authenticator API defined in [I-D.ietf-tls-exported-authenticator] takes as input a request, a set of certificates, and supporting information about the certificate (OCSP, SCT, etc.). The result is an opaque token which is used when generating the "CERTIFICATE" frame.

Upon receipt of a "CERTIFICATE" frame, an endpoint MUST perform the following steps to validate the token it contains:

- o Verify that either the "UNSOLICITED" flag is set (clients only) or that the Request-ID field contains the Request-ID of a previously-sent "CERTIFICATE\_REQUEST" frame.
- o Using the "get context" API, retrieve the "certificate\_request\_context" used to generate the authenticator, if any. Verify that the "certificate\_request\_context" begins with the supplied Request-ID, if any.
- o Use the "validate" API to confirm the validity of the authenticator with regard to the generated request (if any).

If the authenticator cannot be validated, this SHOULD be treated as a connection error of type "CERTIFICATE\_UNREADABLE".

Once the authenticator is accepted, the endpoint can perform any other checks for the acceptability of the certificate itself. Clients MUST NOT accept any end-entity certificate from an exported authenticator which does not contain the Required Domain extension; see Section 5 and Section 6.1.

#### 4. Indicating Failures During HTTP-Layer Certificate Authentication

Because this draft permits certificates to be exchanged at the HTTP framing layer instead of the TLS layer, several certificate-related errors which are defined at the TLS layer might now occur at the HTTP framing layer.

There are two classes of errors which might be encountered, and they are handled differently.

#### 4.1. Misbehavior

This category of errors could indicate a peer failing to follow restrictions in this document, or might indicate that the connection is not fully secure. These errors are fatal to stream or connection, as appropriate.

`CERTIFICATE_OVERUSED` (0xERROR-TBD1): More certificates were used on a request than were requested

`CERTIFICATE_WITHOUT_CONSENT` (0xERROR-TBD2): A `CERTIFICATE_NEEDED` frame was received by a peer which did not indicate support for this extension.

`CERTIFICATE_UNREADABLE` (0xERROR-TBD3): An exported authenticator could not be validated.

#### 4.2. Invalid Certificates

Unacceptable certificates (expired, revoked, or insufficient to satisfy the request) are not treated as stream or connection errors. This is typically not an indication of a protocol failure. Servers SHOULD process requests with the indicated certificate, likely resulting in a "4XX"-series status code in the response. Clients SHOULD establish a new connection in an attempt to reach an authoritative server.

#### 5. Required Domain Certificate Extension

The Required Domain extension allows certificates to limit their use with Secondary Certificate Authentication. A client MUST verify that the server has proven ownership of the indicated identity before accepting the limited certificate over Secondary Certificate Authentication.

The identity in this extension is a restriction asserted by the requester of the certificate and is not verified by the CA. Conforming CAs SHOULD mark the `requiredDomain` extension as non-critical. Conforming CAs MUST require the presence of a CAA record [RFC6844] prior to issuing a certificate with this extension. Because a Required Domain value of "\*" has a much higher risk of reuse if compromised, conforming Certificate Authorities are encouraged to require more extensive verification prior to issuing such a certificate.

The required domain is represented as a `GeneralName`, as specified in Section 4.2.1.6 of [RFC5280]. Unlike the subject field, conforming CAs MUST NOT issue certificates with a `requiredDomain` extension

containing empty GeneralName fields. Clients that encounter such a certificate when processing a certification path MUST consider the certificate invalid.

The wildcard character "\_" MAY be used to represent that any previously authenticated identity is acceptable. This character MUST be the entirety of the name if used and MUST have a type of "dNSName". (That is, "\_" is acceptable, but "\_.com" and "w\_.example.com" are not).

id-ce-requiredDomain OBJECT IDENTIFIER ::= { id-ce TBD1 }

RequiredDomain ::= GeneralName

## 6. Security Considerations

This mechanism defines an alternate way to obtain server and client certificates other than in the initial TLS handshake. While the signature of exported authenticator values is expected to be equally secure, it is important to recognize that a vulnerability in this code path is at least equal to a vulnerability in the TLS handshake.

### 6.1. Impersonation

This mechanism could increase the impact of a key compromise. Rather than needing to subvert DNS or IP routing in order to use a compromised certificate, a malicious server now only needs a client to connect to some HTTPS site under its control in order to present the compromised certificate. Clients SHOULD consult DNS for hostnames presented in secondary certificates if they would have done so for the same hostname if it were present in the primary certificate.

As recommended in [RFC8336], clients opting not to consult DNS ought to employ some alternative means to increase confidence that the certificate is legitimate.

One such means is the Required Domain certificate extension defined in {extension}. Clients MUST require that server certificates presented via this mechanism contain the Required Domain extension and require that a certificate previously accepted on the connection (including the certificate presented in TLS) lists the Required Domain in the Subject field or the Subject Alternative Name extension.

As noted in the Security Considerations of [I-D.ietf-tls-exported-authenticator], it is difficult to formally prove that an endpoint is jointly authoritative over multiple



certificates, rather than individually authoritative on each certificate. As a result, clients MUST NOT assume that because one origin was previously colocated with another, those origins will be reachable via the same endpoints in the future. Clients MUST NOT consider previous secondary certificates to be validated after TLS session resumption. However, clients MAY proactively query for previously-presented secondary certificates.

## 6.2. Fingerprinting

This draft defines a mechanism which could be used to probe servers for origins they support, but opens no new attack versus making repeat TLS connections with different SNI values. Servers SHOULD impose similar denial-of-service mitigations (e.g. request rate limits) to "CERTIFICATE\_REQUEST" frames as to new TLS connections.

While the extensions in the "CERTIFICATE\_REQUEST" frame permit the sender to enumerate the acceptable Certificate Authorities for the requested certificate, it might not be prudent (either for security or data consumption) to include the full list of trusted Certificate Authorities in every request. Senders, particularly clients, SHOULD send only the extensions that narrowly specify which certificates would be acceptable.

Servers can also learn information about clients using this mechanism. The hostnames a user agent finds interesting and retrieves certificates for might indicate origins the user has previously accessed.

## 6.3. Denial of Service

Failure to provide a certificate for a stream after receiving "CERTIFICATE\_NEEDED" blocks processing, and SHOULD be subject to standard timeouts used to guard against unresponsive peers.

Validating a multitude of signatures can be computationally expensive, while generating an invalid signature is computationally cheap. Implementations will require checks for attacks from this direction. Invalid exported authenticators SHOULD be treated as a session error, to avoid further attacks from the peer, though an implementation MAY instead disable HTTP-layer certificates for the current connection instead.

## 6.4. Persistence of Service

CNAME records in the DNS are frequently used to delegate authority for an origin to a third-party provider. This delegation can be

changed without notice, even to the third-party provider, simply by modifying the CNAME record in question.

After the owner of the domain has redirected traffic elsewhere by changing the CNAME, new connections will not arrive for that origin, but connections which are properly directed to this provider for other origins would continue to claim control of this origin (via ORIGIN frame and Secondary Certificates). This is proper behavior based on the third-party provider's configuration, but would likely not be what is intended by the owner of the origin.

This is not an issue which can be mitigated by the protocol, but something about which third-party providers SHOULD educate their customers before using the features described in this document.

#### 6.5. Confusion About State

Implementations need to be aware of the potential for confusion about the state of a connection. The presence or absence of a validated certificate can change during the processing of a request, potentially multiple times, as "USE\_CERTIFICATE" frames are received. A server that uses certificate authentication needs to be prepared to reevaluate the authorization state of a request as the set of certificates changes.

### 7. IANA Considerations

This draft adds entries in three registries.

The feature negotiation settings is registered in Section 7.1. Four frame types are registered in Section 7.2. Six error codes are registered in Section 7.3.

#### 7.1. HTTP/2 Settings

The SETTINGS\_HTTP\_CLIENT\_CERT\_AUTH and SETTINGS\_HTTP\_SERVER\_CERT\_AUTH settings are registered in the "HTTP/2 Settings" registry established in [RFC7540].

Name	Code	Initial Value	Specification
HTTP_CLIENT_CERT_AUTH	0xSETTING-TBD1	0	Section 2.1
HTTP_SERVER_CERT_AUTH	0xSETTING-TBD2	0	Section 2.1

## 7.2. New HTTP/2 Frames

Four new frame types are registered in the "HTTP/2 Frame Types" registry established in [RFC7540]. The entries in the following table are registered by this document.

Frame Type	Code	Specification
CERTIFICATE_NEEDED	0xFRAME-TBD1	Section 3.1
CERTIFICATE_REQUEST	0xFRAME-TBD2	Section 3.3
CERTIFICATE	0xFRAME-TBD3	Section 3.4
USE_CERTIFICATE	0xFRAME-TBD4	Section 3.2

## 7.3. New HTTP/2 Error Codes

Six new error codes are registered in the "HTTP/2 Error Code" registry established in [RFC7540]. The entries in the following table are registered by this document.

Name	Code	Specification
CERTIFICATE_OVERUSED	0xERROR-TBD1	Section 4
CERTIFICATE_WITHOUT_CONSENT	0xERROR-TBD2	Section 4
CERTIFICATE_UNREADABLE	0xERROR-TBD3	Section 4

## 8. References

### 8.1. Normative References

- [I-D.ietf-tls-exported-authenticator]  
Sullivan, N., "Exported Authenticators in TLS", draft-ietf-tls-exported-authenticator-11 (work in progress), December 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6844] Hallam-Baker, P. and R. Stradling, "DNS Certification Authority Authorization (CAA) Resource Record", RFC 6844, DOI 10.17487/RFC6844, January 2013, <<https://www.rfc-editor.org/info/rfc6844>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7540] Belshé, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

## 8.2. Informative References

- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <<https://www.rfc-editor.org/info/rfc7838>>.

[RFC8336] Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", RFC 8336, DOI 10.17487/RFC8336, March 2018, <<https://www.rfc-editor.org/info/rfc8336>>.

### 8.3. URIs

[1] <https://lists.w3.org/Archives/Public/ietf-http-wg/>

[2] <http://httpwg.github.io/>

[3] <https://github.com/httpwg/http-extensions/labels/secondary-certs>

## Appendix A. Change Log

\*RFC Editor's Note:\* Please remove this section prior to publication of a final version of this document.

### A.1. Since draft-ietf-httpbis-http2-secondary-certs-04:

Editorial updates only.

### A.2. Since draft-ietf-httpbis-http2-secondary-certs-03:

- o "CERTIFICATE\_REQUEST" frames contain the Request-ID, which MUST be checked against the "certificate\_request\_context" of the Exported Authenticator Request
- o "CERTIFICATE" frames contain the Request-ID to which they respond, unless the UNSOLICITED flag is set
- o The Required Domain extension is defined for certificates, which must be present for certificates presented by servers

### A.3. Since draft-ietf-httpbis-http2-secondary-certs-02:

Editorial updates only.

### A.4. Since draft-ietf-httpbis-http2-secondary-certs-01:

- o Clients can send "CERTIFICATE\_NEEDED" for stream 0 rather than speculatively reserving a stream for an origin.
- o Use SETTINGS to disable when a TLS-terminating proxy is present (#617, #651)

A.5. Since draft-ietf-httpbis-http2-secondary-certs-00:

- o All frames sent on stream zero; replaced "AUTOMATIC\_USE" on "CERTIFICATE" with "UNSOLICITED" on "USE\_CERTIFICATE". (#482, #566)
- o Use Exported Requests from the TLS Exported Authenticators draft; eliminate facilities for expressing certificate requirements in "CERTIFICATE\_REQUEST" frame. (#481)

A.6. Since draft-bishop-httpbis-http2-additional-certs-05:

- o Adopted as draft-ietf-httpbis-http2-secondary-certs

#### Acknowledgements

Eric Rescorla pointed out several failings in an earlier revision. Andrei Popov contributed to the TLS considerations.

A substantial portion of Mike's work on this draft was supported by Microsoft during his employment there.

#### Authors' Addresses

Mike Bishop  
Akamai

Email: mbishop@evequefou.be

Nick Sullivan  
Cloudflare

Email: nick@cloudflare.com

Martin Thomson  
Mozilla

Email: martin.thomson@gmail.com

HTTP  
Internet-Draft  
Intended status: Standards Track  
Expires: 16 April 2022

M. Nottingham  
Fastly  
P. Sikora  
Google  
13 October 2021

The Proxy-Status HTTP Response Header Field  
draft-ietf-httpbis-proxy-status-08

Abstract

This document defines the Proxy-Status HTTP field to convey the details of intermediary response handling, including generated errors.

Note to Readers

\_RFC EDITOR: please remove this section before publication\_

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

Working Group information can be found at <https://httpwg.org/> (<https://httpwg.org/>); source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions/labels/proxy-status> (<https://github.com/httpwg/http-extensions/labels/proxy-status>).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 16 April 2022.

## Copyright Notice

Copyright (c) 2021 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 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 . . . . .	3
1.1. Notational Conventions . . . . .	4
2. The Proxy-Status HTTP Field . . . . .	4
2.1. Proxy-Status Parameters . . . . .	6
2.1.1. error . . . . .	6
2.1.2. next-hop . . . . .	8
2.1.3. next-protocol . . . . .	8
2.1.4. received-status . . . . .	8
2.1.5. details . . . . .	9
2.2. Defining New Proxy-Status Parameters . . . . .	9
2.3. Proxy Error Types . . . . .	10
2.3.1. DNS Timeout . . . . .	10
2.3.2. DNS Error . . . . .	10
2.3.3. Destination Not Found . . . . .	11
2.3.4. Destination Unavailable . . . . .	11
2.3.5. Destination IP Prohibited . . . . .	11
2.3.6. Destination IP Unroutable . . . . .	12
2.3.7. Connection Refused . . . . .	12
2.3.8. Connection Terminated . . . . .	12
2.3.9. Connection Timeout . . . . .	13
2.3.10. Connection Read Timeout . . . . .	13
2.3.11. Connection Write Timeout . . . . .	13
2.3.12. Connection Limit Reached . . . . .	14
2.3.13. TLS Protocol Error . . . . .	14
2.3.14. TLS Certificate Error . . . . .	14
2.3.15. TLS Alert Received . . . . .	15
2.3.16. HTTP Request Error . . . . .	15
2.3.17. HTTP Request Denied . . . . .	16
2.3.18. HTTP Incomplete Response . . . . .	16
2.3.19. HTTP Response Header Section Too Large . . . . .	16
2.3.20. HTTP Response Header Field Line Too Large . . . . .	17
2.3.21. HTTP Response Body Too Large . . . . .	17



2.3.22. HTTP Response Trailer Section Too Large . . . . .	18
2.3.23. HTTP Response Trailer Field Line Too Large . . . . .	18
2.3.24. HTTP Response Transfer-Coding Error . . . . .	18
2.3.25. HTTP Response Content-Coding Error . . . . .	19
2.3.26. HTTP Response Timeout . . . . .	19
2.3.27. HTTP Upgrade Failed . . . . .	19
2.3.28. HTTP Protocol Error . . . . .	20
2.3.29. Proxy Internal Response . . . . .	20
2.3.30. Proxy Internal Error . . . . .	20
2.3.31. Proxy Configuration Error . . . . .	21
2.3.32. Proxy Loop Detected . . . . .	21
2.4. Defining New Proxy Error Types . . . . .	21
3. IANA Considerations . . . . .	23
4. Security Considerations . . . . .	23
5. References . . . . .	23
5.1. Normative References . . . . .	23
5.2. Informative References . . . . .	24
Authors' Addresses . . . . .	25

## 1. Introduction

HTTP intermediaries (see Section 3.7 of [HTTP]) -- including both forward proxies and gateways (also known as "reverse proxies") -- have become an increasingly significant part of HTTP deployments. In particular, reverse proxies and Content Delivery Networks (CDNs) form part of the critical infrastructure of many Web sites.

Typically, HTTP intermediaries forward requests towards the origin server (inbound) and then forward their responses back to clients (outbound). However, if an error occurs before a response is obtained from an inbound server, the response is often generated by the intermediary itself.

HTTP accommodates these types of errors with a few status codes; for example, 502 Bad Gateway and 504 Gateway Timeout. However, experience has shown that more information is necessary to aid debugging and communicate what's happened to the client. Additionally, intermediaries sometimes want to convey additional information about their handling of a response, even if they did not generate it.

To enable these uses, Section 2 defines a new HTTP response field to allow intermediaries to convey details of their handling of a response. Section 2.1 enumerates the information that can be added to the field by intermediaries, which can be extended as per Section 2.2. Section 2.3 defines a set of error types for use when a proxy encounters an issue when obtaining a response for the request; these can likewise be extended as per Section 2.4.

### 1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification uses Structured Fields [STRUCTURED-FIELDS] to specify syntax and parsing, and ABNF [RFC5234] as a shorthand for that syntax. The terms sf-list, sf-item, sf-string, sf-token, sf-integer and key refer to the structured types defined therein.

Note that in this specification, "proxy" is used to indicate both forward and reverse proxies, otherwise known as gateways. "Next hop" indicates the connection in the direction leading to the origin server for the request.

## 2. The Proxy-Status HTTP Field

The Proxy-Status HTTP response field allows an intermediary to convey additional information about its handling of a response and its associated request. The syntax of this header field conforms to [STRUCTURED-FIELDS].

It is a List ([STRUCTURED-FIELDS], Section 3.1):

Proxy-Status = sf-list

Each member of the list represents an intermediary that has handled the response. The first member of the list represents the intermediary closest to the origin server, and the last member of the list represents the intermediary closest to the user agent.

For example:

Proxy-Status: revproxyl.example.net, ExampleCDN

indicates that this response was handled first by revproxyl.example.net (a reverse proxy adjacent to the origin server) and then ExampleCDN.

Intermediaries determine when it is appropriate to add the Proxy-Status field to a response. Some might decide to append to it to all responses, whereas others might only do so when specifically configured to, or when the request contains a header field that activates a debugging mode.

Each member of the list identifies the intermediary that inserted the value, and MUST have a type of either sf-string or sf-token. Depending on the deployment, this might be a service name (but not a software or hardware product name; e.g., "Example CDN" is appropriate, but "ExampleProxy" is not, because it doesn't identify the deployment), a hostname ("proxy-3.example.com"), an IP address, or a generated string.

Parameters on each member (as per Section 3.1.2 of [STRUCTURED-FIELDS]) convey additional information about that intermediary's handling of the response and its associated request; see Section 2.1. While all of these parameters are OPTIONAL, intermediaries are encouraged to provide as much information as possible (but see Section 4 for security considerations in doing so).

When adding a value to the Proxy-Status field, intermediaries SHOULD preserve the existing members of the field to allow debugging of the entire chain of intermediaries handling the request, unless explicitly configured to remove them (e.g., to prevent internal network details from leaking; see Section 4).

Origin servers MUST NOT generate the Proxy-Status field.

Proxy-Status MAY be sent as a HTTP trailer field. For example, if an intermediary is streaming a response and the inbound connection suddenly terminates, Proxy-Status can only be appended to the trailer section of the outbound message, since the header section has already been sent. However, because it might be silently discarded along the path to the user agent (as is the case for all trailer fields; see Section 6.5 of [HTTP]), Proxy-Status SHOULD NOT be sent as a trailer field unless it is not possible to send it in the header section.

To allow recipients to reconstruct the relative ordering of Proxy-Status members conveyed in trailer fields with those conveyed in header fields, an intermediary MUST NOT send Proxy-Status as a trailer field unless it has also generated a Proxy-Status header field with the same member (although potentially different parameters) in that message.

For example, a proxy identified as 'ThisProxy' that receives a response bearing a header field:

Proxy-Status: SomeOtherProxy

would add its own entry to the header field:

Proxy-Status: SomeOtherProxy, ThisProxy

thus allowing it to append a trailer field:

```
Proxy-Status: ThisProxy; error=read_timeout
```

... which would thereby allow a downstream recipient to understand that processing by 'SomeOtherProxy' occurred before 'ThisProxy'.

A client MAY promote the Proxy-Status trailer field into a header field by following these steps:

1. For each member trailer\_member of the Proxy-Status trailer field value:
  1. Let header\_member be the first (left-most) value of the Proxy-Status header field value, comparing the sf-token or sf-string character-by-character and without consideration of parameters.
  2. If no matching header\_member is found, continue processing the next trailer\_member.
  3. Replace header\_member with trailer\_member in its entirety, including any parameters.
2. Remove the Proxy-Status trailer field, if empty.

## 2.1. Proxy-Status Parameters

This section lists parameters that can be used on the members of the Proxy-Status field. Unrecognised parameters MUST be ignored.

### 2.1.1. error

The error parameter's value is an sf-token that is a Proxy Error Type. When present, it indicates that the intermediary encountered an issue when obtaining this response.

The presence of some Proxy Error Types indicates that the response was generated by the intermediary itself, rather than being forwarded from the origin server. This is the case when, for example, the origin server can't be contacted, so the proxy has to create its own response.

Other Proxy Error Types can be added to (potentially partial) responses that were generated by the origin server or some other inbound server. For example, if the forward connection abruptly closes, an intermediary might add Proxy-Status with an appropriate error as a trailer field.

Proxy Error Types that are registered with a 'Response only generated by intermediaries' value of 'true' indicate that they can only occur on responses generated by the intermediary. If the value is 'false', the response might be generated by the intermediary or an inbound server.

Section 2.3 lists the Proxy Error Types defined in this document; new ones can be defined using the procedure outlined in Section 2.4.

For example:

```
HTTP/1.1 504 Gateway Timeout
Proxy-Status: ExampleCDN; error=connection_timeout
```

indicates that this 504 response was generated by ExampleCDN, due to a connection timeout when going forward.

Or:

```
HTTP/1.1 429 Too Many Requests
Proxy-Status: r34.example.net; error=http_request_error, ExampleCDN
```

indicates that this 429 Too Many Requests response was generated by r34.example.net, not the CDN or the origin.

When sending the error parameter, the most specific Proxy Error Type SHOULD be sent, provided that it accurately represents the error condition. If an appropriate Proxy Error Type is not defined, there are a number of generic error types (e.g., proxy\_internal\_error, http\_protocol\_error) that can be used. If they are not suitable, consider registering a new Proxy Error Type (see Section 2.4).

Each Proxy Error Type has a Recommended HTTP Status Code. When generating a HTTP response containing error, its HTTP status code SHOULD be set to the Recommended HTTP Status Code. However, there may be circumstances (e.g., for backwards compatibility with previous behaviours, a status code has already been sent) when another status code might be used.

Proxy Error Types can also define any number of extra parameters for use with that type. Their use, like all parameters, is optional. As a result, if an extra parameter is used with a Proxy Error Type for which it is not defined, it will be ignored.

### 2.1.2. next-hop

The next-hop parameter's value is an sf-string or sf-token that identifies the intermediary or origin server selected (and used, if contacted) to obtain this response. It might be a hostname, IP address, or alias.

For example:

```
Proxy-Status: cdn.example.org; next-hop=backend.example.org:8001
```

indicates that cdn.example.org used backend.example.org:8001 as the next hop for this request.

### 2.1.3. next-protocol

The next-protocol parameter's value indicates the ALPN protocol identifier [RFC7301] of the protocol used by the intermediary to connect to the next hop when obtaining this response.

The value MUST be either an sf-token or sf-binary, representing a TLS Application-Layer Protocol Negotiation (ALPN) Protocol ID (see <https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml#alpn-protocol-ids> (<https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml#alpn-protocol-ids>)). If the protocol identifier is able to be expressed as an sf-token using ASCII encoding, that form MUST be used.

For example:

```
Proxy-Status: "proxy.example.org"; next-protocol=h2
```

Note that the APLN identifier is being used here to identify the protocol in use; it may or may not have been actually used in the protocol negotiation.

### 2.1.4. received-status

The received-status parameter's value indicates the HTTP status code that the intermediary received from the next hop server when obtaining this response.

The value MUST be an sf-integer.

For example:

```
Proxy-Status: ExampleCDN; received-status=200
```

#### 2.1.5. details

The details parameter's value is an sf-string containing additional information not captured anywhere else. This can include implementation-specific or deployment-specific information.

For example:

```
Proxy-Status: proxy.example.net; error="http_protocol_error";
              details="Malformed response header: space before colon"
```

#### 2.2. Defining New Proxy-Status Parameters

New Proxy-Status Parameters can be defined by registering them in the HTTP Proxy-Status Parameters registry.

Registration requests are reviewed and approved by Expert Review, as per [RFC8126], Section 4.5. A specification document is appreciated, but not required.

The Expert(s) should consider the following factors when evaluating requests:

- \* Community feedback
- \* If the value is sufficiently well-defined
- \* Generic parameters are preferred over vendor-specific, application-specific or deployment-specific values. If a generic value cannot be agreed upon in the community, the parameter's name should be correspondingly specific (e.g., with a prefix that identifies the vendor, application or deployment).
- \* Parameter names should not conflict with registered extra parameters in the Proxy Error Type Registry.

Registration requests should use the following template:

- \* Name: [a name for the Proxy-Status Parameter that matches key]
- \* Description: [a description of the parameter semantics and value]
- \* Reference: [to a specification defining this parameter; optional]

See the registry at <https://iana.org/assignments/http-proxy-status> (<https://iana.org/assignments/http-proxy-status>) for details on where to send registration requests.

### 2.3. Proxy Error Types

This section lists the Proxy Error Types defined by this document. See Section 2.4 for information about defining new Proxy Error Types.

Note that implementations might not produce all Proxy Error Types. The set of types below is designed to map to existing states in implementations, and so may not be applicable to some.

#### 2.3.1. DNS Timeout

- \* Name: dns\_timeout
- \* Description: The intermediary encountered a timeout when trying to find an IP address for the next hop hostname.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 504
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.2. DNS Error

- \* Name: dns\_error
- \* Description: The intermediary encountered a DNS error when trying to find an IP address for the next hop hostname.
- \* Extra Parameters:
  - rcode: A sf-string conveying the DNS RCODE that indicates the error type. See [RFC8499], Section 3.
  - info-code: A sf-integer conveying the Extended DNS Error Code info-code. See [RFC8914].
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: true
- \* Reference: [this document]



### 2.3.3. Destination Not Found

- \* Name: destination\_not\_found
- \* Description: The intermediary cannot determine the appropriate next hop to use for this request; for example, it may not be configured. Note that this error is specific to gateways, which typically require specific configuration to identify the "backend" server; forward proxies use in-band information to identify the origin server.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 500
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

### 2.3.4. Destination Unavailable

- \* Name: destination\_unavailable
- \* Description: The intermediary considers the next hop to be unavailable; e.g., recent attempts to communicate with it may have failed, or a health check may indicate that it is down.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 503
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

### 2.3.5. Destination IP Prohibited

- \* Name: destination\_ip\_prohibited
- \* Description: The intermediary is configured to prohibit connections to the next hop IP address.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: true

- \* Reference: [this document]

#### 2.3.6. Destination IP Unroutable

- \* Name: destination\_ip\_unroutable
- \* Description: The intermediary cannot find a route to the next hop IP address.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.7. Connection Refused

- \* Name: connection\_refused
- \* Description: The intermediary's connection to the next hop was refused.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.8. Connection Terminated

- \* Name: connection\_terminated
- \* Description: The intermediary's connection to the next hop was closed before complete response was received.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.9. Connection Timeout

- \* Name: connection\_timeout
- \* Description: The intermediary's attempt to open a connection to the next hop timed out.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 504
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.10. Connection Read Timeout

- \* Name: connection\_read\_timeout
- \* Description: The intermediary was expecting data on a connection (e.g., part of a response), but did not receive any new data in a configured time limit.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 504
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.11. Connection Write Timeout

- \* Name: connection\_write\_timeout
- \* Description: The intermediary was attempting to write data to a connection, but was not able to (e.g., because its buffers were full).
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 504
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.12. Connection Limit Reached

- \* Name: connection\_limit\_reached
- \* Description: The intermediary is configured to limit the number of connections it has to the next hop, and that limit has been passed.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 503
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.13. TLS Protocol Error

- \* Name: tls\_protocol\_error
- \* Description: The intermediary encountered a TLS error when communicating with the next hop, either during handshake or afterwards.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]
- \* Notes: Not appropriate when a TLS alert is received; see `tls_alert_received`

#### 2.3.14. TLS Certificate Error

- \* Name: tls\_certificate\_error
- \* Description: The intermediary encountered an error when verifying the certificate presented by the next hop.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: true

- \* Reference: [this document]

#### 2.3.15. TLS Alert Received

- \* Name: `tls_alert_received`
- \* Description: The intermediary received a TLS alert from the next hop.
- \* Extra Parameters:
  - `alert-id`: an sf-integer containing the applicable value from the TLS Alerts registry. See [RFC8446].
  - `alert-message`: an sf-token or sf-string containing the applicable description string from the TLS Alerts registry. See [RFC8446].
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.16. HTTP Request Error

- \* Name: `http_request_error`
- \* Description: The intermediary is generating a client (4xx) response on the origin's behalf. Applicable status codes include (but are not limited to) 400, 403, 405, 406, 408, 411, 413, 414, 415, 416, 417, 429.
- \* Extra Parameters:
  - `status-code`: an sf-integer containing the generated status code.
  - `status-phrase`: an sf-string containing the generated status phrase.
- \* Recommended HTTP status code: The applicable 4xx status code
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

- \* Notes: This type helps distinguish between responses generated by intermediaries from those generated by the origin.

#### 2.3.17. HTTP Request Denied

- \* Name: http\_request\_denied
- \* Description: The intermediary rejected the HTTP request based on its configuration and/or policy settings. The request wasn't forwarded to the next hop.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 403
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.18. HTTP Incomplete Response

- \* Name: http\_response\_incomplete
- \* Description: The intermediary received an incomplete response to the request from the next hop.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.19. HTTP Response Header Section Too Large

- \* Name: http\_response\_header\_section\_size
- \* Description: The intermediary received a response to the request whose header section was considered too large.
- \* Extra Parameters:
  - header-section-size: an sf-integer indicating how large the headers received were. Note that they might not be complete; i.e., the intermediary may have discarded or refused additional data.

- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.20. HTTP Response Header Field Line Too Large

- \* Name: http\_response\_header\_size
- \* Description: The intermediary received a response to the request containing an individual header field line that was considered too large.
- \* Extra Parameters:
  - header-name: an sf-string indicating the name of the header field that triggered the error.
  - header-size: an sf-integer indicating the size of the header field that triggered the error.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.21. HTTP Response Body Too Large

- \* Name: http\_response\_body\_size
- \* Description: The intermediary received a response to the request whose body was considered too large.
- \* Extra Parameters:
  - body-size: an sf-integer indicating how large the body received was. Note that it may not have been complete; i.e., the intermediary may have discarded or refused additional data.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

### 2.3.22. HTTP Response Trailer Section Too Large

- \* Name: `http_response_trailer_section_size`
- \* Description: The intermediary received a response to the request whose trailer section was considered too large.
- \* Extra Parameters:
  - `trailer-section-size`: an sf-integer indicating how large the trailers received were. Note that they might not be complete; i.e., the intermediary may have discarded or refused additional data.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

### 2.3.23. HTTP Response Trailer Field Line Too Large

- \* Name: `http_response_trailer_size`
- \* Description: The intermediary received a response to the request containing an individual trailer field line that was considered too large.
- \* Extra Parameters:
  - `trailer-name`: an sf-string indicating the name of the trailer field that triggered the error.
  - `trailer-size`: an sf-integer indicating the size of the trailer field that triggered the error.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

### 2.3.24. HTTP Response Transfer-Coding Error

- \* Name: `http_response_transfer_coding`
- \* Description: The intermediary encountered an error decoding the transfer-coding of the response.



- \* Extra Parameters:
  - coding: an sf-token containing the specific coding (from the HTTP Transfer Coding Registry) that caused the error.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.25. HTTP Response Content-Coding Error

- \* Name: http\_response\_content\_coding
- \* Description: The intermediary encountered an error decoding the content-coding of the response.
- \* Extra Parameters:
  - coding: an sf-token containing the specific coding (from the HTTP Content Coding Registry) that caused the error.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.26. HTTP Response Timeout

- \* Name: http\_response\_timeout
- \* Description: The intermediary reached a configured time limit waiting for the complete response.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 504
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.27. HTTP Upgrade Failed

- \* Name: http\_upgrade\_failed

- \* Description: The HTTP Upgrade between the intermediary and the next hop failed.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.28. HTTP Protocol Error

- \* Name: http\_protocol\_error
- \* Description: The intermediary encountered a HTTP protocol error when communicating with the next hop. This error should only be used when a more specific one is not defined.
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: false
- \* Reference: [this document]

#### 2.3.29. Proxy Internal Response

- \* Name: proxy\_internal\_response
- \* Description: The intermediary generated the response locally, without attempting to connect to the next hop (e.g. in response to a request to a debug endpoint terminated at the intermediary).
- \* Extra Parameters: None.
- \* Recommended HTTP status code: The most appropriate status code for the response
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.30. Proxy Internal Error

- \* Name: proxy\_internal\_error

- \* Description: The intermediary encountered an internal error unrelated to the origin.
- \* Extra Parameters: None
- \* Recommended HTTP status code: 500
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.31. Proxy Configuration Error

- \* Name: proxy\_configuration\_error
- \* Description: The intermediary encountered an error regarding its configuration.
- \* Extra Parameters: None
- \* Recommended HTTP status code: 500
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

#### 2.3.32. Proxy Loop Detected

- \* Name: proxy\_loop\_detected
- \* Description: The intermediary tried to forward the request to itself, or a loop has been detected using different means (e.g. [RFC8586]).
- \* Extra Parameters: None.
- \* Recommended HTTP status code: 502
- \* Response only generated by intermediaries: true
- \* Reference: [this document]

### 2.4. Defining New Proxy Error Types

New Proxy Error Types can be defined by registering them in the HTTP Proxy Error Types registry.

Registration requests are reviewed and approved by Expert Review, as per [RFC8126], Section 4.5. A specification document is appreciated, but not required.

The Expert(s) should consider the following factors when evaluating requests:

- \* Community feedback
- \* If the value is sufficiently well-defined
- \* Generic types are preferred over vendor-specific, application-specific or deployment-specific values. If a generic value cannot be agreed upon in the community, the types's name should be correspondingly specific (e.g., with a prefix that identifies the vendor, application or deployment).
- \* Extra Parameters should not conflict with registered Proxy-Status parameters.

Registration requests should use the following template:

- \* Name: [a name for the Proxy Error Type that matches sf-token]
- \* Description: [a description of the conditions that generate the Proxy Error Type]
- \* Extra Parameters: [zero or more optional parameters, along with their allowable type(s)]
- \* Recommended HTTP status code: [the appropriate HTTP status code for this entry]
- \* Response only generated by intermediaries: ['true' or 'false']
- \* Reference: [to a specification defining this error type; optional]
- \* Notes: [optional]

If the Proxy Error Type might occur in responses that are not generated by the intermediary -- for example, when an error is detected as the response is streamed from a forward connection, causing a Proxy-Status trailer field to be appended -- the 'Response only generated by intermediaries' should be 'false'. If the Proxy Error Type only occurs in responses that are generated by the intermediary, it should be 'true'.

See the registry at <https://iana.org/assignments/http-proxy-status> (<https://iana.org/assignments/http-proxy-status>) for details on where to send registration requests.

### 3. IANA Considerations

Upon publication, please create the HTTP Proxy-Status Parameters registry and the HTTP Proxy Error Types registry at <https://iana.org/assignments/http-proxy-status> (<https://iana.org/assignments/http-proxy-status>) and populate them with the types defined in Section 2.1 and Section 2.3 respectively; see Section 2.2 and Section 2.4 for its associated procedures.

Additionally, please register the following entry in the Hypertext Transfer Protocol (HTTP) Field Name Registry:

- \* Field name: Proxy-Status
- \* Status: permanent
- \* Specification document(s): [this document]
- \* Comments:

### 4. Security Considerations

One of the primary security concerns when using Proxy-Status is leaking information that might aid an attacker. For example, information about the intermediary's configuration and back-end topology can be exposed, allowing attackers to directly target back-end services that are not prepared for high traffic volume or malformed inputs. Some information might only be suitable to reveal to authorized parties.

As a result, care needs to be taken when deciding to generate a Proxy-Status field and what information to include in it. Note that intermediaries are not required to generate a Proxy-Status field in any response, and can conditionally generate them based upon request attributes (e.g., authentication tokens, IP address).

Likewise, generation of all parameters is optional, as is generation of the field itself. Also, the field's content is not verified; an intermediary can claim certain actions (e.g., sending a request over an encrypted channel) but fail to actually do that.

### 5. References

#### 5.1. Normative References

- [HTTP] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>>.
- [STRUCTURED-FIELDS] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/rfc/rfc8941>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.
- [RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <<https://www.rfc-editor.org/rfc/rfc8499>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/rfc/rfc7301>>.
- [RFC8914] Kumari, W., Hunt, E., Arends, R., Hardaker, W., and D. Lawrence, "Extended DNS Errors", RFC 8914, DOI 10.17487/RFC8914, October 2020, <<https://www.rfc-editor.org/rfc/rfc8914>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/rfc/rfc8446>>.

## 5.2. Informative References

- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/rfc/rfc5234>>.
- [RFC8586] Ludin, S., Nottingham, M., and N. Sullivan, "Loop Detection in Content Delivery Networks (CDNs)", RFC 8586, DOI 10.17487/RFC8586, April 2019, <<https://www.rfc-editor.org/rfc/rfc8586>>.

## Authors' Addresses

Mark Nottingham  
Fastly  
Pahran  
Australia

Email: [mnot@mnot.net](mailto:mnot@mnot.net)  
URI: <https://www.mnot.net/>

Piotr Sikora  
Google

Email: [piotrsikora@google.com](mailto:piotrsikora@google.com)

HTTP  
Internet-Draft  
Obsoletes: 6265 (if approved)  
Intended status: Standards Track  
Expires: 26 October 2022

L. Chen, Ed.  
Google LLC  
S. Englehardt, Ed.  
Mozilla  
M. West, Ed.  
Google LLC  
J. Wilander, Ed.  
Apple, Inc  
24 April 2022

Cookies: HTTP State Management Mechanism  
draft-ietf-httpbis-rfc6265bis-10

## Abstract

This document defines the HTTP Cookie and Set-Cookie header fields. These header fields can be used by HTTP servers to store state (called cookies) at HTTP user agents, letting the servers maintain a stateful session over the mostly stateless HTTP protocol. Although cookies have many historical infelicities that degrade their security and privacy, the Cookie and Set-Cookie header fields are widely used on the Internet. This document obsoletes RFC 6265.

## About This Document

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

Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-httpbis-rfc6265bis/>.

Discussion of this document takes place on the HTTP Working Group mailing list (<mailto:ietf-http-wg@w3.org>), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/>. Working Group information can be found at <https://httpwg.org/>.

Source for this draft and an issue tracker can be found at <https://github.com/httpwg/http-extensions/labels/6265bis>.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.



Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 October 2022.

## 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

1. Introduction . . . . .	4
2. Conventions . . . . .	5
2.1. Conformance Criteria . . . . .	5
2.2. Syntax Notation . . . . .	6
2.3. Terminology . . . . .	6
3. Overview . . . . .	7
3.1. Examples . . . . .	8
4. Server Requirements . . . . .	9
4.1. Set-Cookie . . . . .	9
4.1.1. Syntax . . . . .	10
4.1.2. Semantics (Non-Normative) . . . . .	11
4.1.3. Cookie Name Prefixes . . . . .	15
4.2. Cookie . . . . .	16

4.2.1.	Syntax . . . . .	16
4.2.2.	Semantics . . . . .	16
5.	User Agent Requirements . . . . .	17
5.1.	Subcomponent Algorithms . . . . .	17
5.1.1.	Dates . . . . .	17
5.1.2.	Canonicalized Host Names . . . . .	19
5.1.3.	Domain Matching . . . . .	20
5.1.4.	Paths and Path-Match . . . . .	20
5.2.	"Same-site" and "cross-site" Requests . . . . .	21
5.2.1.	Document-based requests . . . . .	21
5.2.2.	Worker-based requests . . . . .	22
5.3.	Ignoring Set-Cookie Header Fields . . . . .	23
5.4.	The Set-Cookie Header Field . . . . .	24
5.4.1.	The Expires Attribute . . . . .	26
5.4.2.	The Max-Age Attribute . . . . .	27
5.4.3.	The Domain Attribute . . . . .	27
5.4.4.	The Path Attribute . . . . .	28
5.4.5.	The Secure Attribute . . . . .	28
5.4.6.	The HttpOnly Attribute . . . . .	28
5.4.7.	The SameSite Attribute . . . . .	28
5.5.	Storage Model . . . . .	30
5.6.	Retrieval Model . . . . .	36
5.6.1.	The Cookie Header Field . . . . .	36
5.6.2.	Non-HTTP APIs . . . . .	36
5.6.3.	Retrieval Algorithm . . . . .	37
6.	Implementation Considerations . . . . .	39
6.1.	Limits . . . . .	39
6.2.	Application Programming Interfaces . . . . .	39
6.3.	IDNA Dependency and Migration . . . . .	40
7.	Privacy Considerations . . . . .	40
7.1.	Third-Party Cookies . . . . .	41
7.2.	Cookie Policy . . . . .	41
7.3.	User Controls . . . . .	42
7.4.	Expiration Dates . . . . .	42
8.	Security Considerations . . . . .	43
8.1.	Overview . . . . .	43
8.2.	Ambient Authority . . . . .	43
8.3.	Clear Text . . . . .	44
8.4.	Session Identifiers . . . . .	44
8.5.	Weak Confidentiality . . . . .	45
8.6.	Weak Integrity . . . . .	46
8.7.	Reliance on DNS . . . . .	47
8.8.	SameSite Cookies . . . . .	47
8.8.1.	Defense in depth . . . . .	47
8.8.2.	Top-level Navigations . . . . .	47
8.8.3.	Mashups and Widgets . . . . .	48
8.8.4.	Server-controlled . . . . .	48
8.8.5.	Reload navigations . . . . .	48

8.8.6. Top-level requests with "unsafe" methods . . . . .	49
9. IANA Considerations . . . . .	50
9.1. Cookie . . . . .	50
9.2. Set-Cookie . . . . .	50
9.3. Cookie Attribute Registry . . . . .	50
9.3.1. Procedure . . . . .	51
9.3.2. Registration . . . . .	51
10. References . . . . .	51
10.1. Normative References . . . . .	51
10.2. Informative References . . . . .	53
Appendix A. Changes . . . . .	55
A.1. draft-ietf-httpbis-rfc6265bis-00 . . . . .	55
A.2. draft-ietf-httpbis-rfc6265bis-01 . . . . .	55
A.3. draft-ietf-httpbis-rfc6265bis-02 . . . . .	56
A.4. draft-ietf-httpbis-rfc6265bis-03 . . . . .	56
A.5. draft-ietf-httpbis-rfc6265bis-04 . . . . .	57
A.6. draft-ietf-httpbis-rfc6265bis-05 . . . . .	57
A.7. draft-ietf-httpbis-rfc6265bis-06 . . . . .	57
A.8. draft-ietf-httpbis-rfc6265bis-07 . . . . .	58
A.9. draft-ietf-httpbis-rfc6265bis-08 . . . . .	58
A.10. draft-ietf-httpbis-rfc6265bis-09 . . . . .	59
A.11. draft-ietf-httpbis-rfc6265bis-10 . . . . .	59
Acknowledgements . . . . .	60
Authors' Addresses . . . . .	60

## 1. Introduction

This document defines the HTTP Cookie and Set-Cookie header fields. Using the Set-Cookie header field, an HTTP server can pass name/value pairs and associated metadata (called cookies) to a user agent. When the user agent makes subsequent requests to the server, the user agent uses the metadata and other information to determine whether to return the name/value pairs in the Cookie header field.

Although simple on their surface, cookies have a number of complexities. For example, the server indicates a scope for each cookie when sending it to the user agent. The scope indicates the maximum amount of time in which the user agent should return the cookie, the servers to which the user agent should return the cookie, and the URI schemes for which the cookie is applicable.

For historical reasons, cookies contain a number of security and privacy infelicities. For example, a server can indicate that a given cookie is intended for "secure" connections, but the Secure attribute does not provide integrity in the presence of an active network attacker. Similarly, cookies for a given host are shared across all the ports on that host, even though the usual "same-origin policy" used by web browsers isolates content retrieved via different ports.

There are two audiences for this specification: developers of cookie-generating servers and developers of cookie-consuming user agents.

To maximize interoperability with user agents, servers SHOULD limit themselves to the well-behaved profile defined in Section 4 when generating cookies.

User agents MUST implement the more liberal processing rules defined in Section 5, in order to maximize interoperability with existing servers that do not conform to the well-behaved profile defined in Section 4.

This document specifies the syntax and semantics of these header fields as they are actually used on the Internet. In particular, this document does not create new syntax or semantics beyond those in use today. The recommendations for cookie generation provided in Section 4 represent a preferred subset of current server behavior, and even the more liberal cookie processing algorithm provided in Section 5 does not recommend all of the syntactic and semantic variations in use today. Where some existing software differs from the recommended protocol in significant ways, the document contains a note explaining the difference.

This document obsoletes [RFC6265].

## 2. Conventions

### 2.1. Conformance Criteria

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("MUST", "SHOULD", "MAY", etc.) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps can be implemented in any manner, so long as the end result is equivalent. In particular, the algorithms defined in this specification are intended to be easy to understand and are not intended to be performant.

## 2.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

The following core rules are included by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTLs (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), NUL (null octet), OCTET (any 8-bit sequence of data except NUL), SP (space), HTAB (horizontal tab), CHAR (any [USASCII] character), VCHAR (any visible [USASCII] character), and WSP (whitespace).

The OWS (optional whitespace) and BWS (bad whitespace) rules are defined in Section 5.6.3 of [HTTPSEM].

## 2.3. Terminology

The terms "user agent", "client", "server", "proxy", and "origin server" have the same meaning as in the HTTP/1.1 specification ([HTTPSEM], Section 3).

The request-host is the name of the host, as known by the user agent, to which the user agent is sending an HTTP request or from which it is receiving an HTTP response (i.e., the name of the host to which it sent the corresponding HTTP request).

The term request-uri refers to "target URI" as defined in Section 7.1 of [HTTPSEM].

Two sequences of octets are said to case-insensitively match each other if and only if they are equivalent under the i;ascii-casemap collation defined in [RFC4790].

The term string means a sequence of non-NUL octets.

The terms "active document", "ancestor browsing context", "browsing context", "dedicated worker", "Document", "nested browsing context", "opaque origin", "parent browsing context", "sandboxed origin browsing context flag", "shared worker", "the worker's Documents", "top-level browsing context", and "WorkerGlobalScope" are defined in [HTML].

"Service Workers" are defined in the Service Workers specification [SERVICE-WORKERS].

The term "origin", the mechanism of deriving an origin from a URI, and the "the same" matching algorithm for origins are defined in [RFC6454].

"Safe" HTTP methods include GET, HEAD, OPTIONS, and TRACE, as defined in Section 9.2.1 of [HTTPSEM].

A domain's "public suffix" is the portion of a domain that is controlled by a public registry, such as "com", "co.uk", and "pvt.k12.wy.us". A domain's "registrable domain" is the domain's public suffix plus the label to its left. That is, for `https://www.site.example`, the public suffix is `example`, and the registrable domain is `site.example`. Whenever possible, user agents SHOULD use an up-to-date public suffix list, such as the one maintained by the Mozilla project at [PSL].

The term "request", as well as a request's "client", "current url", "method", "target browsing context", and "url list", are defined in [FETCH].

The term "non-HTTP APIs" refers to non-HTTP mechanisms used to set and retrieve cookies, such as a web browser API that exposes cookies to scripts.

### 3. Overview

This section outlines a way for an origin server to send state information to a user agent and for the user agent to return the state information to the origin server.

To store state, the origin server includes a Set-Cookie header field in an HTTP response. In subsequent requests, the user agent returns a Cookie request header field to the origin server. The Cookie header field contains cookies the user agent received in previous Set-Cookie header fields. The origin server is free to ignore the Cookie header field or use its contents for an application-defined purpose.

Origin servers MAY send a Set-Cookie response header field with any response. An origin server can include multiple Set-Cookie header fields in a single response. The presence of a Cookie or a Set-Cookie header field does not preclude HTTP caches from storing and reusing a response.

Origin servers SHOULD NOT fold multiple Set-Cookie header fields into a single header field. The usual mechanism for folding HTTP headers fields (i.e., as defined in Section 5.3 of [HTTPSEM]) might change the semantics of the Set-Cookie header field because the %x2C (",") character is used by Set-Cookie in a way that conflicts with such folding.

User agents MAY ignore Set-Cookie header fields based on response status codes or the user agent's cookie policy (see Section 5.3).

### 3.1. Examples

Using the Set-Cookie header field, a server can send the user agent a short string in an HTTP response that the user agent will return in future HTTP requests that are within the scope of the cookie. For example, the server can send the user agent a "session identifier" named SID with the value 31d4d96e407aad42. The user agent then returns the session identifier in subsequent requests.

```
== Server -> User Agent ==
```

```
Set-Cookie: SID=31d4d96e407aad42
```

```
== User Agent -> Server ==
```

```
Cookie: SID=31d4d96e407aad42
```

The server can alter the default scope of the cookie using the Path and Domain attributes. For example, the server can instruct the user agent to return the cookie to every path and every subdomain of site.example.

```
== Server -> User Agent ==
```

```
Set-Cookie: SID=31d4d96e407aad42; Path=/; Domain=site.example
```

```
== User Agent -> Server ==
```

```
Cookie: SID=31d4d96e407aad42
```

As shown in the next example, the server can store multiple cookies at the user agent. For example, the server can store a session identifier as well as the user's preferred language by returning two Set-Cookie header fields. Notice that the server uses the Secure and HttpOnly attributes to provide additional security protections for the more sensitive session identifier (see Section 4.1.2).

== Server -> User Agent ==

```
Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
Set-Cookie: lang=en-US; Path=/; Domain=site.example
```

== User Agent -> Server ==

```
Cookie: SID=31d4d96e407aad42; lang=en-US
```

Notice that the Cookie header field above contains two cookies, one named SID and one named lang. If the server wishes the user agent to persist the cookie over multiple "sessions" (e.g., user agent restarts), the server can specify an expiration date in the Expires attribute. Note that the user agent might delete the cookie before the expiration date if the user agent's cookie store exceeds its quota or if the user manually deletes the server's cookie.

== Server -> User Agent ==

```
Set-Cookie: lang=en-US; Expires=Wed, 09 Jun 2021 10:18:14 GMT
```

== User Agent -> Server ==

```
Cookie: SID=31d4d96e407aad42; lang=en-US
```

Finally, to remove a cookie, the server returns a Set-Cookie header field with an expiration date in the past. The server will be successful in removing the cookie only if the Path and the Domain attribute in the Set-Cookie header field match the values used when the cookie was created.

== Server -> User Agent ==

```
Set-Cookie: lang=; Expires=Sun, 06 Nov 1994 08:49:37 GMT
```

== User Agent -> Server ==

```
Cookie: SID=31d4d96e407aad42
```

#### 4. Server Requirements

This section describes the syntax and semantics of a well-behaved profile of the Cookie and Set-Cookie header fields.

##### 4.1. Set-Cookie

The Set-Cookie HTTP response header field is used to send cookies from the server to the user agent.



## 4.1.1. Syntax

Informally, the Set-Cookie response header field contains a cookie, which begins with a name-value-pair, followed by zero or more attribute-value pairs. Servers SHOULD NOT send Set-Cookie header fields that fail to conform to the following grammar:

```

set-cookie           = set-cookie-string
set-cookie-string    = BWS cookie-pair *( BWS ";" OWS cookie-av )
cookie-pair          = cookie-name BWS "=" BWS cookie-value
cookie-name          = 1*cookie-octet
cookie-value         = *cookie-octet / ( DQUOTE *cookie-octet DQUOTE )
cookie-octet         = %x21 / %x23-2B / %x2D-3A / %x3C-5B / %x5D-7E
                      / %x80-FF
                      ; octets excluding CTLs,
                      ; whitespace DQUOTE, comma, semicolon,
                      ; and backslash

cookie-av            = expires-av / max-age-av / domain-av /
                      path-av / secure-av / httponly-av /
                      samesite-av / extension-av
expires-av           = "Expires" BWS "=" BWS sane-cookie-date
sane-cookie-date     =
    <IMF-fixdate, defined in [HTTPSEM], Section 5.6.7>
max-age-av           = "Max-Age" BWS "=" BWS non-zero-digit *DIGIT
non-zero-digit       = %x31-39
                      ; digits 1 through 9
domain-av            = "Domain" BWS "=" BWS domain-value
domain-value         = <subdomain>
                      ; see details below
path-av              = "Path" BWS "=" BWS path-value
path-value           = *av-octet
secure-av            = "Secure"
httponly-av          = "HttpOnly"
samesite-av          = "SameSite" BWS "=" BWS samesite-value
samesite-value       = "Strict" / "Lax" / "None"
extension-av         = *av-octet
av-octet             = %x20-3A / %x3C-7E
                      ; any CHAR except CTLs or ";"

```

Note that some of the grammatical terms above reference documents that use different grammatical notations than this document (which uses ABNF from [RFC5234]).

The semantics of the cookie-value are not defined by this document.

To maximize compatibility with user agents, servers that wish to store arbitrary data in a cookie-value SHOULD encode that data, for example, using Base64 [RFC4648].

The domain-value is a subdomain as defined by [RFC1034], Section 3.5, and as enhanced by [RFC1123], Section 2.1. Thus, domain-value is a string of [USASCII] characters, such as one obtained by applying the "ToASCII" operation defined in Section 4 of [RFC3490].

Per the grammar above, the cookie-value MAY be wrapped in DQUOTE characters. Note that in this case, the initial and trailing DQUOTE characters are not stripped. They are part of the cookie-value, and will be included in Cookie header fields sent to the server.

The portions of the set-cookie-string produced by the cookie-av term are known as attributes. To maximize compatibility with user agents, servers SHOULD NOT produce two attributes with the same name in the same set-cookie-string. (See Section 5.5 for how user agents handle this case.)

Servers SHOULD NOT include more than one Set-Cookie header field in the same response with the same cookie-name. (See Section 5.4 for how user agents handle this case.)

If a server sends multiple responses containing Set-Cookie header fields concurrently to the user agent (e.g., when communicating with the user agent over multiple sockets), these responses create a "race condition" that can lead to unpredictable behavior.

NOTE: Some existing user agents differ in their interpretation of two-digit years. To avoid compatibility issues, servers SHOULD use the rfc1123-date format, which requires a four-digit year.

NOTE: Some user agents store and process dates in cookies as 32-bit UNIX time\_t values. Implementation bugs in the libraries supporting time\_t processing on some systems might cause such user agents to process dates after the year 2038 incorrectly.

#### 4.1.2. Semantics (Non-Normative)

This section describes simplified semantics of the Set-Cookie header field. These semantics are detailed enough to be useful for understanding the most common uses of cookies by servers. The full semantics are described in Section 5.

When the user agent receives a Set-Cookie header field, the user agent stores the cookie together with its attributes. Subsequently, when the user agent makes an HTTP request, the user agent includes the applicable, non-expired cookies in the Cookie header field.

If the user agent receives a new cookie with the same cookie-name, domain-value, and path-value as a cookie that it has already stored, the existing cookie is evicted and replaced with the new cookie. Notice that servers can delete cookies by sending the user agent a new cookie with an Expires attribute with a value in the past.

Unless the cookie's attributes indicate otherwise, the cookie is returned only to the origin server (and not, for example, to any subdomains), and it expires at the end of the current session (as defined by the user agent). User agents ignore unrecognized cookie attributes (but not the entire cookie).

#### 4.1.2.1. The Expires Attribute

The Expires attribute indicates the maximum lifetime of the cookie, represented as the date and time at which the cookie expires. The user agent is not required to retain the cookie until the specified date has passed. In fact, user agents often evict cookies due to memory pressure or privacy concerns.

The user agent MUST limit the maximum value of the Expires attribute. The limit SHOULD NOT be greater than 400 days (34560000 seconds) in the future. The RECOMMENDED limit is 400 days in the future, but the user agent MAY adjust the limit (see Section 7.2). Expires attributes that are greater than the limit MUST be reduced to the limit.

#### 4.1.2.2. The Max-Age Attribute

The Max-Age attribute indicates the maximum lifetime of the cookie, represented as the number of seconds until the cookie expires. The user agent is not required to retain the cookie for the specified duration. In fact, user agents often evict cookies due to memory pressure or privacy concerns.

The user agent MUST limit the maximum value of the Max-Age attribute. The limit SHOULD NOT be greater than 400 days (34560000 seconds) in duration. The RECOMMENDED limit is 400 days in duration, but the user agent MAY adjust the limit (see Section 7.2). Max-Age attributes that are greater than the limit MUST be reduced to the limit.

NOTE: Some existing user agents do not support the Max-Age attribute. User agents that do not support the Max-Age attribute ignore the attribute.

If a cookie has both the Max-Age and the Expires attribute, the Max-Age attribute has precedence and controls the expiration date of the cookie. If a cookie has neither the Max-Age nor the Expires attribute, the user agent will retain the cookie until "the current session is over" (as defined by the user agent).

#### 4.1.2.3. The Domain Attribute

The Domain attribute specifies those hosts to which the cookie will be sent. For example, if the value of the Domain attribute is "site.example", the user agent will include the cookie in the Cookie header field when making HTTP requests to site.example, www.site.example, and www.corp.site.example. (Note that a leading %x2E ("."), if present, is ignored even though that character is not permitted, but a trailing %x2E ("."), if present, will cause the user agent to ignore the attribute.) If the server omits the Domain attribute, the user agent will return the cookie only to the origin server.

WARNING: Some existing user agents treat an absent Domain attribute as if the Domain attribute were present and contained the current host name. For example, if site.example returns a Set-Cookie header field without a Domain attribute, these user agents will erroneously send the cookie to www.site.example as well.

The user agent will reject cookies unless the Domain attribute specifies a scope for the cookie that would include the origin server. For example, the user agent will accept a cookie with a Domain attribute of "site.example" or of "foo.site.example" from foo.site.example, but the user agent will not accept a cookie with a Domain attribute of "bar.site.example" or of "baz.foo.site.example".

NOTE: For security reasons, many user agents are configured to reject Domain attributes that correspond to "public suffixes". For example, some user agents will reject Domain attributes of "com" or "co.uk". (See Section 5.5 for more information.)

#### 4.1.2.4. The Path Attribute

The scope of each cookie is limited to a set of paths, controlled by the Path attribute. If the server omits the Path attribute, the user agent will use the "directory" of the request-uri's path component as the default value. (See Section 5.1.4 for more details.)

The user agent will include the cookie in an HTTP request only if the path portion of the request-uri matches (or is a subdirectory of) the cookie's Path attribute, where the %x2F ("/") character is interpreted as a directory separator.

Although seemingly useful for isolating cookies between different paths within a given host, the Path attribute cannot be relied upon for security (see Section 8).

#### 4.1.2.5. The Secure Attribute

The Secure attribute limits the scope of the cookie to "secure" channels (where "secure" is defined by the user agent). When a cookie has the Secure attribute, the user agent will include the cookie in an HTTP request only if the request is transmitted over a secure channel (typically HTTP over Transport Layer Security (TLS) [RFC2818]).

#### 4.1.2.6. The HttpOnly Attribute

The HttpOnly attribute limits the scope of the cookie to HTTP requests. In particular, the attribute instructs the user agent to omit the cookie when providing access to cookies via non-HTTP APIs.

Note that the HttpOnly attribute is independent of the Secure attribute: a cookie can have both the HttpOnly and the Secure attribute.

#### 4.1.2.7. The SameSite Attribute

The "SameSite" attribute limits the scope of the cookie such that it will only be attached to requests if those requests are same-site, as defined by the algorithm in Section 5.2. For example, requests for `https://site.example/sekrit-image` will attach same-site cookies if and only if initiated from a context whose "site for cookies" is an origin with a scheme and registered domain of "https" and "site.example" respectively.

If the "SameSite" attribute's value is "Strict", the cookie will only be sent along with "same-site" requests. If the value is "Lax", the cookie will be sent with same-site requests, and with "cross-site" top-level navigations, as described in Section 5.4.7.1. If the value is "None", the cookie will be sent with same-site and cross-site requests. If the "SameSite" attribute's value is something other than these three known keywords, the attribute's value will be subject to a default enforcement mode that is equivalent to "Lax".

The "SameSite" attribute affects cookie creation as well as delivery. Cookies which assert "SameSite=Lax" or "SameSite=Strict" cannot be set in responses to cross-site subresource requests, or cross-site nested navigations. They can be set along with any top-level navigation, cross-site or otherwise.

#### 4.1.3. Cookie Name Prefixes

Section 8.5 and Section 8.6 of this document spell out some of the drawbacks of cookies' historical implementation. In particular, it is impossible for a server to have confidence that a given cookie was set with a particular set of attributes. In order to provide such confidence in a backwards-compatible way, two common sets of requirements can be inferred from the first few characters of the cookie's name.

The normative requirements for the prefixes described below are detailed in the storage model algorithm defined in Section 5.5.

##### 4.1.3.1. The "\_\_Secure-" Prefix

If a cookie's name begins with a case-sensitive match for the string `__Secure-`, then the cookie will have been set with a Secure attribute.

For example, the following Set-Cookie header field would be rejected by a conformant user agent, as it does not have a Secure attribute.

```
Set-Cookie: __Secure-SID=12345; Domain=site.example
```

Whereas the following Set-Cookie header field would be accepted if set from a secure origin (e.g. "https://site.example/"), and rejected otherwise:

```
Set-Cookie: __Secure-SID=12345; Domain=site.example; Secure
```

##### 4.1.3.2. The "\_\_Host-" Prefix

If a cookie's name begins with a case-sensitive match for the string `__Host-`, then the cookie will have been set with a Secure attribute, a Path attribute with a value of `/`, and no Domain attribute.

This combination yields a cookie that hews as closely as a cookie can to treating the origin as a security boundary. The lack of a Domain attribute ensures that the cookie's host-only-flag is true, locking the cookie to a particular host, rather than allowing it to span subdomains. Setting the Path to / means that the cookie is effective for the entire host, and won't be overridden for specific paths. The Secure attribute ensures that the cookie is unaltered by non-secure origins, and won't span protocols.

Ports are the only piece of the origin model that \_\_Host- cookies continue to ignore.

For example, the following cookies would always be rejected:

```
Set-Cookie: __Host-SID=12345
Set-Cookie: __Host-SID=12345; Secure
Set-Cookie: __Host-SID=12345; Domain=site.example
Set-Cookie: __Host-SID=12345; Domain=site.example; Path=/
Set-Cookie: __Host-SID=12345; Secure; Domain=site.example; Path=/
```

While the following would be accepted if set from a secure origin (e.g. "https://site.example/"), and rejected otherwise:

```
Set-Cookie: __Host-SID=12345; Secure; Path=/
```

## 4.2. Cookie

### 4.2.1. Syntax

The user agent sends stored cookies to the origin server in the Cookie header field. If the server conforms to the requirements in Section 4.1 (and the user agent conforms to the requirements in Section 5), the user agent will send a Cookie header field that conforms to the following grammar:

```
cookie          = cookie-string
cookie-string   = cookie-pair *( ";" SP cookie-pair )
```

### 4.2.2. Semantics

Each cookie-pair represents a cookie stored by the user agent. The cookie-pair contains the cookie-name and cookie-value the user agent received in the Set-Cookie header field.

Notice that the cookie attributes are not returned. In particular, the server cannot determine from the Cookie field alone when a cookie will expire, for which hosts the cookie is valid, for which paths the cookie is valid, or whether the cookie was set with the Secure or HttpOnly attributes.

The semantics of individual cookies in the Cookie header field are not defined by this document. Servers are expected to imbue these cookies with application-specific semantics.

Although cookies are serialized linearly in the Cookie header field, servers SHOULD NOT rely upon the serialization order. In particular, if the Cookie header field contains two cookies with the same name (e.g., that were set with different Path or Domain attributes), servers SHOULD NOT rely upon the order in which these cookies appear in the header field.

## 5. User Agent Requirements

This section specifies the Cookie and Set-Cookie header fields in sufficient detail that a user agent implementing these requirements precisely can interoperate with existing servers (even those that do not conform to the well-behaved profile described in Section 4).

A user agent could enforce more restrictions than those specified herein (e.g., restrictions specified by its cookie policy, described in Section 7.2). However, such additional restrictions may reduce the likelihood that a user agent will be able to interoperate with existing servers.

### 5.1. Subcomponent Algorithms

This section defines some algorithms used by user agents to process specific subcomponents of the Cookie and Set-Cookie header fields.

#### 5.1.1. Dates

The user agent MUST use an algorithm equivalent to the following algorithm to parse a cookie-date. Note that the various boolean flags defined as a part of the algorithm (i.e., found-time, found-day-of-month, found-month, found-year) are initially "not set".

1. Using the grammar below, divide the cookie-date into date-tokens.



```

cookie-date      = *delimiter date-token-list *delimiter
date-token-list = date-token *( 1*delimiter date-token )
date-token       = 1*non-delimiter

delimiter        = %x09 / %x20-2F / %x3B-40 / %x5B-60 / %x7B-7E
non-delimiter     = %x00-08 / %x0A-1F / DIGIT / ":" / ALPHA
                  / %x7F-FF
non-digit        = %x00-2F / %x3A-FF

day-of-month     = 1*2DIGIT [ non-digit *OCTET ]
month            = ( "jan" / "feb" / "mar" / "apr" /
                  "may" / "jun" / "jul" / "aug" /
                  "sep" / "oct" / "nov" / "dec" ) *OCTET
year            = 2*4DIGIT [ non-digit *OCTET ]
time            = hms-time [ non-digit *OCTET ]
hms-time        = time-field ":" time-field ":" time-field
time-field      = 1*2DIGIT

```

2. Process each date-token sequentially in the order the date-tokens appear in the cookie-date:
  1. If the found-time flag is not set and the token matches the time production, set the found-time flag and set the hour-value, minute-value, and second-value to the numbers denoted by the digits in the date-token, respectively. Skip the remaining sub-steps and continue to the next date-token.
  2. If the found-day-of-month flag is not set and the date-token matches the day-of-month production, set the found-day-of-month flag and set the day-of-month-value to the number denoted by the date-token. Skip the remaining sub-steps and continue to the next date-token.
  3. If the found-month flag is not set and the date-token matches the month production, set the found-month flag and set the month-value to the month denoted by the date-token. Skip the remaining sub-steps and continue to the next date-token.
  4. If the found-year flag is not set and the date-token matches the year production, set the found-year flag and set the year-value to the number denoted by the date-token. Skip the remaining sub-steps and continue to the next date-token.
3. If the year-value is greater than or equal to 70 and less than or equal to 99, increment the year-value by 1900.
4. If the year-value is greater than or equal to 0 and less than or equal to 69, increment the year-value by 2000.

1. NOTE: Some existing user agents interpret two-digit years differently.
5. Abort these steps and fail to parse the cookie-date if:
  - \* at least one of the found-day-of-month, found-month, found-year, or found-time flags is not set,
  - \* the day-of-month-value is less than 1 or greater than 31,
  - \* the year-value is less than 1601,
  - \* the hour-value is greater than 23,
  - \* the minute-value is greater than 59, or
  - \* the second-value is greater than 59.

(Note that leap seconds cannot be represented in this syntax.)
6. Let the parsed-cookie-date be the date whose day-of-month, month, year, hour, minute, and second (in UTC) are the day-of-month-value, the month-value, the year-value, the hour-value, the minute-value, and the second-value, respectively. If no such date exists, abort these steps and fail to parse the cookie-date.
7. Return the parsed-cookie-date as the result of this algorithm.

#### 5.1.2. Canonicalized Host Names

A canonicalized host name is the string generated by the following algorithm:

1. Convert the host name to a sequence of individual domain name labels.
2. Convert each label that is not a Non-Reserved LDH (NR-LDH) label, to an A-label (see Section 2.3.2.1 of [RFC5890] for the former and latter), or to a "punycode label" (a label resulting from the "ToASCII" conversion in Section 4 of [RFC3490]), as appropriate (see Section 6.3 of this specification).
3. Concatenate the resulting labels, separated by a %x2E (".") character.

### 5.1.3. Domain Matching

A string domain-matches a given domain string if at least one of the following conditions hold:

- \* The domain string and the string are identical. (Note that both the domain string and the string will have been canonicalized to lower case at this point.)
- \* All of the following conditions hold:
  - The domain string is a suffix of the string.
  - The last character of the string that is not included in the domain string is a %x2E (".") character.
  - The string is a host name (i.e., not an IP address).

### 5.1.4. Paths and Path-Match

The user agent MUST use an algorithm equivalent to the following algorithm to compute the default-path of a cookie:

1. Let uri-path be the path portion of the request-uri if such a portion exists (and empty otherwise).
2. If the uri-path is empty or if the first character of the uri-path is not a %x2F ("/") character, output %x2F ("/") and skip the remaining steps.
3. If the uri-path contains no more than one %x2F ("/") character, output %x2F ("/") and skip the remaining step.
4. Output the characters of the uri-path from the first character up to, but not including, the right-most %x2F ("/").

A request-path path-matches a given cookie-path if at least one of the following conditions holds:

- \* The cookie-path and the request-path are identical.

Note that this differs from the rules in [RFC3986] for equivalence of the path component, and hence two equivalent paths can have different cookies.

- \* The cookie-path is a prefix of the request-path, and the last character of the cookie-path is %x2F ("/").

- \* The cookie-path is a prefix of the request-path, and the first character of the request-path that is not included in the cookie-path is a %x2F ("/") character.

## 5.2. "Same-site" and "cross-site" Requests

Two origins are same-site if they satisfy the "same site" criteria defined in [SAMESITE]. A request is "same-site" if the following criteria are true:

1. The request is not the result of a cross-site redirect. That is, the origin of every url in the request's url list is same-site with the request's current url's origin.
2. The request is not the result of a reload navigation triggered through a user interface element (as defined by the user agent; e.g., a request triggered by the user clicking a refresh button on a toolbar).
3. The request's current url's origin is same-site with the request's client's "site for cookies" (which is an origin), or if the request has no client or the request's client is null.

Requests which are the result of a reload navigation triggered through a user interface element are same-site if the reloaded document was originally navigated to via a same-site request. A request that is not "same-site" is instead "cross-site".

The request's client's "site for cookies" is calculated depending upon its client's type, as described in the following subsections:

### 5.2.1. Document-based requests

The URI displayed in a user agent's address bar is the only security context directly exposed to users, and therefore the only signal users can reasonably rely upon to determine whether or not they trust a particular website. The origin of that URI represents the context in which a user most likely believes themselves to be interacting. We'll define this origin, the top-level browsing context's active document's origin, as the "top-level origin".

For a document displayed in a top-level browsing context, we can stop here: the document's "site for cookies" is the top-level origin.

For documents which are displayed in nested browsing contexts, we need to audit the origins of each of a document's ancestor browsing contexts' active documents in order to account for the "multiple-nested scenarios" described in Section 4 of [RFC7034]. A document's

"site for cookies" is the top-level origin if and only if the top-level origin is same-site with the document's origin, and with each of the document's ancestor documents' origins. Otherwise its "site for cookies" is an origin set to an opaque origin.

Given a Document (document), the following algorithm returns its "site for cookies":

1. Let top-document be the active document in document's browsing context's top-level browsing context.
2. Let top-origin be the origin of top-document's URI if top-document's sandboxed origin browsing context flag is set, and top-document's origin otherwise.
3. Let documents be a list containing document and each of document's ancestor browsing contexts' active documents.
4. For each item in documents:
  1. Let origin be the origin of item's URI if item's sandboxed origin browsing context flag is set, and item's origin otherwise.
  2. If origin is not same-site with top-origin, return an origin set to an opaque origin.
5. Return top-origin.

#### 5.2.2. Worker-based requests

Worker-driven requests aren't as clear-cut as document-driven requests, as there isn't a clear link between a top-level browsing context and a worker. This is especially true for Service Workers [SERVICE-WORKERS], which may execute code in the background, without any document visible at all.

Note: The descriptions below assume that workers must be same-origin with the documents that instantiate them. If this invariant changes, we'll need to take the worker's script's URI into account when determining their status.

##### 5.2.2.1. Dedicated and Shared Workers

Dedicated workers are simple, as each dedicated worker is bound to one and only one document. Requests generated from a dedicated worker (via `importScripts`, `XMLHttpRequest`, `fetch()`, etc) define their "site for cookies" as that document's "site for cookies".

Shared workers may be bound to multiple documents at once. As it is quite possible for those documents to have distinct "site for cookies" values, the worker's "site for cookies" will be an origin set to an opaque origin in cases where the values are not all same-site with the worker's origin, and the worker's origin in cases where the values agree.

Given a `WorkerGlobalScope` (`worker`), the following algorithm returns its "site for cookies":

1. Let `site` be `worker`'s origin.
2. For each document in `worker`'s Documents:
  1. Let `document-site` be document's "site for cookies" (as defined in Section 5.2.1).
  2. If `document-site` is not same-site with `site`, return an origin set to an opaque origin.
3. Return `site`.

#### 5.2.2.2. Service Workers

Service Workers are more complicated, as they act as a completely separate execution context with only tangential relationship to the Document which registered them.

Requests which simply pass through a Service Worker will be handled as described above: the request's client will be the Document or Worker which initiated the request, and its "site for cookies" will be those defined in Section 5.2.1 and Section 5.2.2.1

Requests which are initiated by the Service Worker itself (via a direct call to `fetch()`, for instance), on the other hand, will have a client which is a `ServiceWorkerGlobalScope`. Its "site for cookies" will be the Service Worker's URI's origin.

Given a `ServiceWorkerGlobalScope` (`worker`), the following algorithm returns its "site for cookies":

1. Return `worker`'s origin.

#### 5.3. Ignoring Set-Cookie Header Fields

User agents MAY ignore Set-Cookie header fields contained in responses with 100-level status codes or based on its cookie policy (see Section 7.2).

All other Set-Cookie header fields SHOULD be processed according to Section 5.4. That is, Set-Cookie header fields contained in responses with non-100-level status codes (including those in responses with 400- and 500-level status codes) SHOULD be processed unless ignored according to the user agent's cookie policy.

#### 5.4. The Set-Cookie Header Field

When a user agent receives a Set-Cookie header field in an HTTP response, the user agent MAY ignore the Set-Cookie header field in its entirety (see Section 5.3).

If the user agent does not ignore the Set-Cookie header field in its entirety, the user agent MUST parse the field-value of the Set-Cookie header field as a set-cookie-string (defined below).

NOTE: The algorithm below is more permissive than the grammar in Section 4.1. For example, the algorithm strips leading and trailing whitespace from the cookie name and value (but maintains internal whitespace), whereas the grammar in Section 4.1 forbids whitespace in these positions. In addition, the algorithm below accommodates some characters that are not cookie-octets according to the grammar in Section 4.1. User agents use this algorithm so as to interoperate with servers that do not follow the recommendations in Section 4.

NOTE: As set-cookie-string may originate from a non-HTTP API, it is not guaranteed to be free of CTL characters, so this algorithm handles them explicitly. Horizontal tab (%x09) is excluded from the CTL characters that lead to set-cookie-string rejection, as it is considered whitespace, which is handled separately.

NOTE: The set-cookie-string may contain octet sequences that appear percent-encoded as per Section 2.1 of [RFC3986]. However, a user agent MUST NOT decode these sequences and instead parse the individual octets as specified in this algorithm.

A user agent MUST use an algorithm equivalent to the following algorithm to parse a set-cookie-string:

1. If the set-cookie-string contains a %x00-08 / %x0A-1F / %x7F character (CTL characters excluding HTAB): Abort these steps and ignore the set-cookie-string entirely.
2. If the set-cookie-string contains a %x3B (";") character:

1. The name-value-pair string consists of the characters up to, but not including, the first %x3B (";"), and the unparsed-attributes consist of the remainder of the set-cookie-string (including the %x3B (";") in question).

Otherwise:

1. The name-value-pair string consists of all the characters contained in the set-cookie-string, and the unparsed-attributes is the empty string.
3. If the name-value-pair string lacks a %x3D ("=") character, then the name string is empty, and the value string is the value of name-value-pair.

Otherwise, the name string consists of the characters up to, but not including, the first %x3D ("=") character, and the (possibly empty) value string consists of the characters after the first %x3D ("=") character.

4. Remove any leading or trailing WSP characters from the name string and the value string.
5. If the sum of the lengths of the name string and the value string is more than 4096 octets, abort these steps and ignore the set-cookie-string entirely.
6. The cookie-name is the name string, and the cookie-value is the value string.

The user agent MUST use an algorithm equivalent to the following algorithm to parse the unparsed-attributes:

1. If the unparsed-attributes string is empty, skip the rest of these steps.
2. Discard the first character of the unparsed-attributes (which will be a %x3B (";") character).
3. If the remaining unparsed-attributes contains a %x3B (";") character:
  1. Consume the characters of the unparsed-attributes up to, but not including, the first %x3B (";") character.

Otherwise:

1. Consume the remainder of the unparsed-attributes.



Let the cookie-av string be the characters consumed in this step.

4. If the cookie-av string contains a %x3D ("=") character:
  1. The (possibly empty) attribute-name string consists of the characters up to, but not including, the first %x3D ("=") character, and the (possibly empty) attribute-value string consists of the characters after the first %x3D ("=") character.
- Otherwise:
  1. The attribute-name string consists of the entire cookie-av string, and the attribute-value string is empty.
5. Remove any leading or trailing WSP characters from the attribute-name string and the attribute-value string.
6. If the attribute-value is longer than 1024 octets, ignore the cookie-av string and return to Step 1 of this algorithm.
7. Process the attribute-name and attribute-value according to the requirements in the following subsections. (Notice that attributes with unrecognized attribute-names are ignored.)
8. Return to Step 1 of this algorithm.

When the user agent finishes parsing the set-cookie-string, the user agent is said to "receive a cookie" from the request-uri with name cookie-name, value cookie-value, and attributes cookie-attribute-list. (See Section 5.5 for additional requirements triggered by receiving a cookie.)

#### 5.4.1. The Expires Attribute

If the attribute-name case-insensitively matches the string "Expires", the user agent MUST process the cookie-av as follows.

1. Let the expiry-time be the result of parsing the attribute-value as cookie-date (see Section 5.1.1).
2. If the attribute-value failed to parse as a cookie date, ignore the cookie-av.
3. Let cookie-age-limit be the maximum age of the cookie (which SHOULD be 400 days in the future or sooner, see Section 4.1.2.1).

4. If the expiry-time is more than cookie-age-limit, the user agent MUST set the expiry time to cookie-age-limit in seconds.
5. If the expiry-time is earlier than the earliest date the user agent can represent, the user agent MAY replace the expiry-time with the earliest representable date.
6. Append an attribute to the cookie-attribute-list with an attribute-name of Expires and an attribute-value of expiry-time.

#### 5.4.2. The Max-Age Attribute

If the attribute-name case-insensitively matches the string "Max-Age", the user agent MUST process the cookie-av as follows.

1. If the first character of the attribute-value is not a DIGIT or a "-" character, ignore the cookie-av.
2. If the remainder of attribute-value contains a non-DIGIT character, ignore the cookie-av.
3. Let delta-seconds be the attribute-value converted to an integer.
4. Let cookie-age-limit be the maximum age of the cookie (which SHOULD be 400 days or less, see Section 4.1.2.2).
5. Set delta-seconds to the smaller of its present value and cookie-age-limit.
6. If delta-seconds is less than or equal to zero (0), let expiry-time be the earliest representable date and time. Otherwise, let the expiry-time be the current date and time plus delta-seconds seconds.
7. Append an attribute to the cookie-attribute-list with an attribute-name of Max-Age and an attribute-value of expiry-time.

#### 5.4.3. The Domain Attribute

If the attribute-name case-insensitively matches the string "Domain", the user agent MUST process the cookie-av as follows.

1. Let cookie-domain be the attribute-value.
2. If cookie-domain starts with %x2E ("."), let cookie-domain be cookie-domain without its leading %x2E (".").
3. Convert the cookie-domain to lower case.

4. Append an attribute to the cookie-attribute-list with an attribute-name of Domain and an attribute-value of cookie-domain.

#### 5.4.4. The Path Attribute

If the attribute-name case-insensitively matches the string "Path", the user agent MUST process the cookie-av as follows.

1. If the attribute-value is empty or if the first character of the attribute-value is not %x2F ("/"):

1. Let cookie-path be the default-path.

Otherwise:

1. Let cookie-path be the attribute-value.

2. Append an attribute to the cookie-attribute-list with an attribute-name of Path and an attribute-value of cookie-path.

#### 5.4.5. The Secure Attribute

If the attribute-name case-insensitively matches the string "Secure", the user agent MUST append an attribute to the cookie-attribute-list with an attribute-name of Secure and an empty attribute-value.

#### 5.4.6. The HttpOnly Attribute

If the attribute-name case-insensitively matches the string "HttpOnly", the user agent MUST append an attribute to the cookie-attribute-list with an attribute-name of HttpOnly and an empty attribute-value.

#### 5.4.7. The SameSite Attribute

If the attribute-name case-insensitively matches the string "SameSite", the user agent MUST process the cookie-av as follows:

1. Let enforcement be "Default".
2. If cookie-av's attribute-value is a case-insensitive match for "None", set enforcement to "None".
3. If cookie-av's attribute-value is a case-insensitive match for "Strict", set enforcement to "Strict".
4. If cookie-av's attribute-value is a case-insensitive match for "Lax", set enforcement to "Lax".

5. Append an attribute to the cookie-attribute-list with an attribute-name of "SameSite" and an attribute-value of enforcement.

#### 5.4.7.1. "Strict" and "Lax" enforcement

Same-site cookies in "Strict" enforcement mode will not be sent along with top-level navigations which are triggered from a cross-site document context. As discussed in Section 8.8.2, this might or might not be compatible with existing session management systems. In the interests of providing a drop-in mechanism that mitigates the risk of CSRF attacks, developers may set the SameSite attribute in a "Lax" enforcement mode that carves out an exception which sends same-site cookies along with cross-site requests if and only if they are top-level navigations which use a "safe" (in the [HTTPSEM] sense) HTTP method. (Note that a request's method may be changed from POST to GET for some redirects (see Sections 15.4.2 and 15.4.3 of [HTTPSEM]); in these cases, a request's "safe"ness is determined based on the method of the current redirect hop.)

Lax enforcement provides reasonable defense in depth against CSRF attacks that rely on unsafe HTTP methods (like POST), but does not offer a robust defense against CSRF as a general category of attack:

1. Attackers can still pop up new windows or trigger top-level navigations in order to create a "same-site" request (as described in Section 5.2.1), which is only a speedbump along the road to exploitation.
2. Features like <link rel='prerender'> [prerendering] can be exploited to create "same-site" requests without the risk of user detection.

When possible, developers should use a session management mechanism such as that described in Section 8.8.2 to mitigate the risk of CSRF more completely.

#### 5.4.7.2. "Lax-Allowing-Unsafe" enforcement

As discussed in Section 8.8.6, compatibility concerns may necessitate the use of a "Lax-allowing-unsafe" enforcement mode that allows cookies to be sent with a cross-site HTTP request if and only if it is a top-level request, regardless of request method. That is, the "Lax-allowing-unsafe" enforcement mode waives the requirement for the HTTP request's method to be "safe" in the SameSite enforcement step of the retrieval algorithm in Section 5.6.3. (All cookies, regardless of SameSite enforcement mode, may be set for top-level navigations, regardless of HTTP request method, as specified in

Section 5.5.)

"Lax-allowing-unsafe" is not a distinct value of the SameSite attribute. Rather, user agents MAY apply "Lax-allowing-unsafe" enforcement only to cookies that did not explicitly specify a SameSite attribute (i.e., those whose same-site-flag was set to "Default" by default). To limit the scope of this compatibility mode, user agents which apply "Lax-allowing-unsafe" enforcement SHOULD restrict the enforcement to cookies which were created recently. Deployment experience has shown a cookie age of 2 minutes or less to be a reasonable limit.

If the user agent uses "Lax-allowing-unsafe" enforcement, it MUST apply the following modification to the retrieval algorithm defined in Section 5.6.3:

Replace the condition in the penultimate bullet point of step 1 of the retrieval algorithm reading

- \* The HTTP request associated with the retrieval uses a "safe" method.

with

- \* At least one of the following is true:
  1. The HTTP request associated with the retrieval uses a "safe" method.
  2. The cookie's same-site-flag is "Default" and the amount of time elapsed since the cookie's creation-time is at most a duration of the user agent's choosing.

## 5.5. Storage Model

The user agent stores the following fields about each cookie: name, value, expiry-time, domain, path, creation-time, last-access-time, persistent-flag, host-only-flag, secure-only-flag, http-only-flag, and same-site-flag.

When the user agent "receives a cookie" from a request-uri with name cookie-name, value cookie-value, and attributes cookie-attribute-list, the user agent MUST process the cookie as follows:

1. A user agent MAY ignore a received cookie in its entirety. See Section 5.3.

2. If cookie-name is empty and cookie-value is empty, abort these steps and ignore the cookie entirely.
3. If the cookie-name or the cookie-value contains a %x00-08 / %x0A-1F / %x7F character (CTL characters excluding HTAB), abort these steps and ignore the cookie entirely.
4. If the sum of the lengths of cookie-name and cookie-value is more than 4096 octets, abort these steps and ignore the cookie entirely.
5. Create a new cookie with name cookie-name, value cookie-value. Set the creation-time and the last-access-time to the current date and time.
6. If the cookie-attribute-list contains an attribute with an attribute-name of "Max-Age":
  1. Set the cookie's persistent-flag to true.
  2. Set the cookie's expiry-time to attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "Max-Age".

Otherwise, if the cookie-attribute-list contains an attribute with an attribute-name of "Expires" (and does not contain an attribute with an attribute-name of "Max-Age"):

1. Set the cookie's persistent-flag to true.
2. Set the cookie's expiry-time to attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "Expires".

Otherwise:

1. Set the cookie's persistent-flag to false.
  2. Set the cookie's expiry-time to the latest representable date.
7. If the cookie-attribute-list contains an attribute with an attribute-name of "Domain":
    1. Let the domain-attribute be the attribute-value of the last attribute in the cookie-attribute-list with both an attribute-name of "Domain" and an attribute-value whose length is no more than 1024 octets. (Note that a leading

%x2E ("."), if present, is ignored even though that character is not permitted, but a trailing %x2E ("."), if present, will cause the user agent to ignore the attribute.)

Otherwise:

1. Let the domain-attribute be the empty string.
8. If the domain-attribute contains a character that is not in the range of [USASCII] characters, abort these steps and ignore the cookie entirely.
9. If the user agent is configured to reject "public suffixes" and the domain-attribute is a public suffix:

1. If the domain-attribute is identical to the canonicalized request-host:

1. Let the domain-attribute be the empty string.

Otherwise:

1. Abort these steps and ignore the cookie entirely.

NOTE: This step prevents attacker.example from disrupting the integrity of site.example by setting a cookie with a Domain attribute of "example".

10. If the domain-attribute is non-empty:
  1. If the canonicalized request-host does not domain-match the domain-attribute:
    1. Abort these steps and ignore the cookie entirely.
  - Otherwise:
    1. Set the cookie's host-only-flag to false.
    2. Set the cookie's domain to the domain-attribute.

Otherwise:

1. Set the cookie's host-only-flag to true.
2. Set the cookie's domain to the canonicalized request-host.

11. If the cookie-attribute-list contains an attribute with an attribute-name of "Path", set the cookie's path to attribute-value of the last attribute in the cookie-attribute-list with both an attribute-name of "Path" and an attribute-value whose length is no more than 1024 octets. Otherwise, set the cookie's path to the default-path of the request-uri.
12. If the cookie-attribute-list contains an attribute with an attribute-name of "Secure", set the cookie's secure-only-flag to true. Otherwise, set the cookie's secure-only-flag to false.
13. If the scheme component of the request-uri does not denote a "secure" protocol (as defined by the user agent), and the cookie's secure-only-flag is true, then abort these steps and ignore the cookie entirely.
14. If the cookie-attribute-list contains an attribute with an attribute-name of "HttpOnly", set the cookie's http-only-flag to true. Otherwise, set the cookie's http-only-flag to false.
15. If the cookie was received from a "non-HTTP" API and the cookie's http-only-flag is true, abort these steps and ignore the cookie entirely.
16. If the cookie's secure-only-flag is false, and the scheme component of request-uri does not denote a "secure" protocol, then abort these steps and ignore the cookie entirely if the cookie store contains one or more cookies that meet all of the following criteria:
  1. Their name matches the name of the newly-created cookie.
  2. Their secure-only-flag is true.
  3. Their domain domain-matches the domain of the newly-created cookie, or vice-versa.
  4. The path of the newly-created cookie path-matches the path of the existing cookie.

Note: The path comparison is not symmetric, ensuring only that a newly-created, non-secure cookie does not overlay an existing secure cookie, providing some mitigation against cookie-fixing attacks. That is, given an existing secure cookie named 'a' with a path of '/login', a non-secure cookie named 'a' could be set for a path of '/' or '/foo', but not for a path of '/login' or '/login/en'.



17. If the cookie-attribute-list contains an attribute with an attribute-name of "SameSite", and an attribute-value of "Strict", "Lax", or "None", set the cookie's same-site-flag to the attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "SameSite". Otherwise, set the cookie's same-site-flag to "Default".
18. If the cookie's same-site-flag is not "None":
  1. If the cookie was received from a "non-HTTP" API, and the API was called from a browsing context's active document whose "site for cookies" is not same-site with the top-level origin, then abort these steps and ignore the newly created cookie entirely.
  2. If the cookie was received from a "same-site" request (as defined in Section 5.2), skip the remaining substeps and continue processing the cookie.
  3. If the cookie was received from a request which is navigating a top-level browsing context [HTML] (e.g. if the request's "reserved client" is either null or an environment whose "target browsing context" is a top-level browsing context), skip the remaining substeps and continue processing the cookie.

Note: Top-level navigations can create a cookie with any SameSite value, even if the new cookie wouldn't have been sent along with the request had it already existed prior to the navigation.
  4. Abort these steps and ignore the newly created cookie entirely.
19. If the cookie's "same-site-flag" is "None", abort these steps and ignore the cookie entirely unless the cookie's secure-only-flag is true.
20. If the cookie-name begins with a case-sensitive match for the string "\_\_Secure-", abort these steps and ignore the cookie entirely unless the cookie's secure-only-flag is true.
21. If the cookie-name begins with a case-sensitive match for the string "\_\_Host-", abort these steps and ignore the cookie entirely unless the cookie meets all the following criteria:
  1. The cookie's secure-only-flag is true.

2. The cookie's host-only-flag is true.
  3. The cookie-attribute-list contains an attribute with an attribute-name of "Path", and the cookie's path is /.
22. If the cookie store contains a cookie with the same name, domain, host-only-flag, and path as the newly-created cookie:
1. Let old-cookie be the existing cookie with the same name, domain, host-only-flag, and path as the newly-created cookie. (Notice that this algorithm maintains the invariant that there is at most one such cookie.)
  2. If the newly-created cookie was received from a "non-HTTP" API and the old-cookie's http-only-flag is true, abort these steps and ignore the newly created cookie entirely.
  3. Update the creation-time of the newly-created cookie to match the creation-time of the old-cookie.
  4. Remove the old-cookie from the cookie store.
23. Insert the newly-created cookie into the cookie store.

A cookie is "expired" if the cookie has an expiry date in the past.

The user agent MUST evict all expired cookies from the cookie store if, at any time, an expired cookie exists in the cookie store.

At any time, the user agent MAY "remove excess cookies" from the cookie store if the number of cookies sharing a domain field exceeds some implementation-defined upper bound (such as 50 cookies).

At any time, the user agent MAY "remove excess cookies" from the cookie store if the cookie store exceeds some predetermined upper bound (such as 3000 cookies).

When the user agent removes excess cookies from the cookie store, the user agent MUST evict cookies in the following priority order:

1. Expired cookies.
2. Cookies whose secure-only-flag is false, and which share a domain field with more than a predetermined number of other cookies.
3. Cookies that share a domain field with more than a predetermined number of other cookies.

#### 4. All cookies.

If two cookies have the same removal priority, the user agent MUST evict the cookie with the earliest last-access-time first.

When "the current session is over" (as defined by the user agent), the user agent MUST remove from the cookie store all cookies with the persistent-flag set to false.

#### 5.6. Retrieval Model

This section defines how cookies are retrieved from a cookie store in the form of a cookie-string. A "retrieval" is any event which requires generating a cookie-string. For example, a retrieval may occur in order to build a Cookie header field for an HTTP request, or may be required in order to return a cookie-string from a call to a "non-HTTP" API that provides access to cookies. A retrieval has an associated URI, same-site status, and type, which are defined below depending on the type of retrieval.

##### 5.6.1. The Cookie Header Field

The user agent includes stored cookies in the Cookie HTTP request header field.

When the user agent generates an HTTP request, the user agent MUST NOT attach more than one Cookie header field.

A user agent MAY omit the Cookie header field in its entirety. For example, the user agent might wish to block sending cookies during "third-party" requests from setting cookies (see Section 7.1).

If the user agent does attach a Cookie header field to an HTTP request, the user agent MUST compute the cookie-string following the algorithm defined in Section 5.6.3, where the retrieval's URI is the request-uri, the retrieval's same-site status is computed for the HTTP request as defined in Section 5.2, and the retrieval's type is "HTTP".

##### 5.6.2. Non-HTTP APIs

The user agent MAY implement "non-HTTP" APIs that can be used to access stored cookies.

A user agent MAY return an empty cookie-string in certain contexts, such as when a retrieval occurs within a third-party context (see Section 7.1).

If a user agent does return cookies for a given call to a "non-HTTP" API with an associated Document, then the user agent MUST compute the cookie-string following the algorithm defined in Section 5.6.3, where the retrieval's URI is defined by the caller (see [DOM-DOCUMENT-COOKIE]), the retrieval's same-site status is "same-site" if the Document's "site for cookies" is same-site with the top-level origin as defined in Section 5.2.1 (otherwise it is "cross-site"), and the retrieval's type is "non-HTTP".

### 5.6.3. Retrieval Algorithm

Given a cookie store and a retrieval, the following algorithm returns a cookie-string from a given cookie store.

1. Let cookie-list be the set of cookies from the cookie store that meets all of the following requirements:

- \* Either:

- The cookie's host-only-flag is true and the canonicalized host of the retrieval's URI is identical to the cookie's domain.

Or:

- The cookie's host-only-flag is false and the canonicalized host of the retrieval's URI domain-matches the cookie's domain.

- \* The retrieval's URI's path path-matches the cookie's path.

- \* If the cookie's secure-only-flag is true, then the retrieval's URI's scheme must denote a "secure" protocol (as defined by the user agent).

NOTE: The notion of a "secure" protocol is not defined by this document. Typically, user agents consider a protocol secure if the protocol makes use of transport-layer security, such as SSL or TLS. For example, most user agents consider "https" to be a scheme that denotes a secure protocol.

- \* If the cookie's http-only-flag is true, then exclude the cookie if the retrieval's type is "non-HTTP".
- \* If the cookie's same-site-flag is not "None" and the retrieval's same-site status is "cross-site", then exclude the cookie unless all of the following conditions are met:

- The retrieval's type is "HTTP".
  - The same-site-flag is "Lax" or "Default".
  - The HTTP request associated with the retrieval uses a "safe" method.
  - The target browsing context of the HTTP request associated with the retrieval is a top-level browsing context.
2. The user agent SHOULD sort the cookie-list in the following order:
- \* Cookies with longer paths are listed before cookies with shorter paths.
  - \* Among cookies that have equal-length path fields, cookies with earlier creation-times are listed before cookies with later creation-times.
- NOTE: Not all user agents sort the cookie-list in this order, but this order reflects common practice when this document was written, and, historically, there have been servers that (erroneously) depended on this order.
3. Update the last-access-time of each cookie in the cookie-list to the current date and time.
4. Serialize the cookie-list into a cookie-string by processing each cookie in the cookie-list in order:
1. If the cookies' name is not empty, output the cookie's name followed by the %x3D ("=") character.
  2. If the cookies' value is not empty, output the cookie's value.
  3. If there is an unprocessed cookie in the cookie-list, output the characters %x3B and %x20 ("; ").

NOTE: Despite its name, the cookie-string is actually a sequence of octets, not a sequence of characters. To convert the cookie-string (or components thereof) into a sequence of characters (e.g., for presentation to the user), the user agent might wish to try using the UTF-8 character encoding [RFC3629] to decode the octet sequence. This decoding might fail, however, because not every sequence of octets is valid UTF-8.

## 6. Implementation Considerations

### 6.1. Limits

Practical user agent implementations have limits on the number and size of cookies that they can store. General-use user agents SHOULD provide each of the following minimum capabilities:

- \* At least 50 cookies per domain.
- \* At least 3000 cookies total.

User agents MAY limit the maximum number of cookies they store, and may evict any cookie at any time (whether at the request of the user or due to implementation limitations).

Note that a limit on the maximum number of cookies also limits the total size of the stored cookies, due to the length limits which MUST be enforced in Section 5.4.

Servers SHOULD use as few and as small cookies as possible to avoid reaching these implementation limits and to minimize network bandwidth due to the Cookie header field being included in every request.

Servers SHOULD gracefully degrade if the user agent fails to return one or more cookies in the Cookie header field because the user agent might evict any cookie at any time.

### 6.2. Application Programming Interfaces

One reason the Cookie and Set-Cookie header fields use such esoteric syntax is that many platforms (both in servers and user agents) provide a string-based application programming interface (API) to cookies, requiring application-layer programmers to generate and parse the syntax used by the Cookie and Set-Cookie header fields, which many programmers have done incorrectly, resulting in interoperability problems.

Instead of providing string-based APIs to cookies, platforms would be well-served by providing more semantic APIs. It is beyond the scope of this document to recommend specific API designs, but there are clear benefits to accepting an abstract "Date" object instead of a serialized date string.

### 6.3. IDNA Dependency and Migration

IDNA2008 [RFC5890] supersedes IDNA2003 [RFC3490]. However, there are differences between the two specifications, and thus there can be differences in processing (e.g., converting) domain name labels that have been registered under one from those registered under the other. There will be a transition period of some time during which IDNA2003-based domain name labels will exist in the wild. User agents SHOULD implement IDNA2008 [RFC5890] and MAY implement [UTS46] or [RFC5895] in order to facilitate their IDNA transition. If a user agent does not implement IDNA2008, the user agent MUST implement IDNA2003 [RFC3490].

## 7. Privacy Considerations

Cookies' primary privacy risk is their ability to correlate user activity. This can happen on a single site, but is most problematic when activity is tracked across different, seemingly unconnected Web sites to build a user profile.

Over time, this capability (warned against explicitly in [RFC2109] and all of its successors) has become widely used for varied reasons including:

- \* authenticating users across sites,
- \* assembling information on users,
- \* protecting against fraud and other forms of undesirable traffic,
- \* targeting advertisements at specific users or at users with specified attributes,
- \* measuring how often ads are shown to users, and
- \* recognizing when an ad resulted in a change in user behavior.

While not every use of cookies is necessarily problematic for privacy, their potential for abuse has become a widespread concern in the Internet community and broader society. In response to these concerns, user agents have actively constrained cookie functionality in various ways (as allowed and encouraged by previous specifications), while avoiding disruption to features they judge desirable for the health of the Web.

It is too early to declare consensus on which specific mechanism(s) should be used to mitigate cookies' privacy impact; user agents' ongoing changes to how they are handled are best characterised as experiments that can provide input into that eventual consensus.

Instead, this document describes limited, general mitigations against the privacy risks associated with cookies that enjoy wide deployment at the time of writing. It is expected that implementations will continue to experiment and impose stricter, more well-defined limitations on cookies over time. Future versions of this document might codify those mechanisms based upon deployment experience. If functions that currently rely on cookies can be supported by separate, targeted mechanisms, they might be documented in separate specifications and stricter limitations on cookies might become feasible.

Note that cookies are not the only mechanism that can be used to track users across sites, so while these mitigations are necessary to improve Web privacy, they are not sufficient on their own.

### 7.1. Third-Party Cookies

A "third-party" or cross-site cookie is one that is associated with embedded content (such as scripts, images, stylesheets, frames) that is obtained from a different server than the one that hosts the primary resource (usually, the Web page that the user is viewing). Third-party cookies are often used to correlate users' activity on different sites.

Because of their inherent privacy issues, most user agents now limit third-party cookies in a variety of ways. Some completely block third-party cookies by refusing to process third-party Set-Cookie header fields and refusing to send third-party Cookie header fields. Some partition cookies based upon the first-party context, so that different cookies are sent depending on the site being browsed. Some block cookies based upon user agent cookie policy and/or user controls.

While this document does not endorse or require a specific approach, it is RECOMMENDED that user agents adopt a policy for third-party cookies that is as restrictive as compatibility constraints permit. Consequently, resources cannot rely upon third-party cookies being treated consistently by user agents for the foreseeable future.

### 7.2. Cookie Policy

User agents MAY enforce a cookie policy consisting of restrictions on how cookies may be used or ignored (see Section 5.3).



A cookie policy may govern which domains or parties, as in first and third parties (see Section 7.1), for which the user agent will allow cookie access. The policy can also define limits on cookie size, cookie expiry (see Section 4.1.2.1 and Section 4.1.2.2), and the number of cookies per domain or in total.

The recommended cookie expiry upper limit is 400 days. User agents may set a lower limit to enforce shorter data retention timelines, or set the limit higher to support longer retention when appropriate (e.g., server-to-server communication over HTTPS).

The goal of a restrictive cookie policy is often to improve security or privacy. User agents often allow users to change the cookie policy (see Section 7.3).

### 7.3. User Controls

User agents SHOULD provide users with a mechanism for managing the cookies stored in the cookie store. For example, a user agent might let users delete all cookies received during a specified time period or all the cookies related to a particular domain. In addition, many user agents include a user interface element that lets users examine the cookies stored in their cookie store.

User agents SHOULD provide users with a mechanism for disabling cookies. When cookies are disabled, the user agent MUST NOT include a Cookie header field in outbound HTTP requests and the user agent MUST NOT process Set-Cookie header fields in inbound HTTP responses.

User agents MAY offer a way to change the cookie policy (see Section 7.2).

User agents MAY provide users the option of preventing persistent storage of cookies across sessions. When configured thusly, user agents MUST treat all received cookies as if the persistent-flag were set to false. Some popular user agents expose this functionality via "private browsing" mode [Aggarwal2010].

### 7.4. Expiration Dates

Although servers can set the expiration date for cookies to the distant future, most user agents do not actually retain cookies for multiple decades. Rather than choosing gratuitously long expiration periods, servers SHOULD promote user privacy by selecting reasonable cookie expiration periods based on the purpose of the cookie. For example, a typical session identifier might reasonably be set to expire in two weeks.

## 8. Security Considerations

### 8.1. Overview

Cookies have a number of security pitfalls. This section overviews a few of the more salient issues.

In particular, cookies encourage developers to rely on ambient authority for authentication, often becoming vulnerable to attacks such as cross-site request forgery [CSRF]. Also, when storing session identifiers in cookies, developers often create session fixation vulnerabilities.

Transport-layer encryption, such as that employed in HTTPS, is insufficient to prevent a network attacker from obtaining or altering a victim's cookies because the cookie protocol itself has various vulnerabilities (see "Weak Confidentiality" and "Weak Integrity", below). In addition, by default, cookies do not provide confidentiality or integrity from network attackers, even when used in conjunction with HTTPS.

### 8.2. Ambient Authority

A server that uses cookies to authenticate users can suffer security vulnerabilities because some user agents let remote parties issue HTTP requests from the user agent (e.g., via HTTP redirects or HTML forms). When issuing those requests, user agents attach cookies even if the remote party does not know the contents of the cookies, potentially letting the remote party exercise authority at an unwary server.

Although this security concern goes by a number of names (e.g., cross-site request forgery, confused deputy), the issue stems from cookies being a form of ambient authority. Cookies encourage server operators to separate designation (in the form of URLs) from authorization (in the form of cookies). Consequently, the user agent might supply the authorization for a resource designated by the attacker, possibly causing the server or its clients to undertake actions designated by the attacker as though they were authorized by the user.

Instead of using cookies for authorization, server operators might wish to consider entangling designation and authorization by treating URLs as capabilities. Instead of storing secrets in cookies, this approach stores secrets in URLs, requiring the remote entity to supply the secret itself. Although this approach is not a panacea, judicious application of these principles can lead to more robust security.

### 8.3. Clear Text

Unless sent over a secure channel (such as TLS), the information in the Cookie and Set-Cookie header fields is transmitted in the clear.

1. All sensitive information conveyed in these header fields is exposed to an eavesdropper.
2. A malicious intermediary could alter the header fields as they travel in either direction, with unpredictable results.
3. A malicious client could alter the Cookie header fields before transmission, with unpredictable results.

Servers SHOULD encrypt and sign the contents of cookies (using whatever format the server desires) when transmitting them to the user agent (even when sending the cookies over a secure channel). However, encrypting and signing cookie contents does not prevent an attacker from transplanting a cookie from one user agent to another or from replaying the cookie at a later time.

In addition to encrypting and signing the contents of every cookie, servers that require a higher level of security SHOULD use the Cookie and Set-Cookie header fields only over a secure channel. When using cookies over a secure channel, servers SHOULD set the Secure attribute (see Section 4.1.2.5) for every cookie. If a server does not set the Secure attribute, the protection provided by the secure channel will be largely moot.

For example, consider a webmail server that stores a session identifier in a cookie and is typically accessed over HTTPS. If the server does not set the Secure attribute on its cookies, an active network attacker can intercept any outbound HTTP request from the user agent and redirect that request to the webmail server over HTTP. Even if the webmail server is not listening for HTTP connections, the user agent will still include cookies in the request. The active network attacker can intercept these cookies, replay them against the server, and learn the contents of the user's email. If, instead, the server had set the Secure attribute on its cookies, the user agent would not have included the cookies in the clear-text request.

### 8.4. Session Identifiers

Instead of storing session information directly in a cookie (where it might be exposed to or replayed by an attacker), servers commonly store a nonce (or "session identifier") in a cookie. When the server receives an HTTP request with a nonce, the server can look up state information associated with the cookie using the nonce as a key.

Using session identifier cookies limits the damage an attacker can cause if the attacker learns the contents of a cookie because the nonce is useful only for interacting with the server (unlike non-nonce cookie content, which might itself be sensitive). Furthermore, using a single nonce prevents an attacker from "splicing" together cookie content from two interactions with the server, which could cause the server to behave unexpectedly.

Using session identifiers is not without risk. For example, the server SHOULD take care to avoid "session fixation" vulnerabilities. A session fixation attack proceeds in three steps. First, the attacker transplants a session identifier from his or her user agent to the victim's user agent. Second, the victim uses that session identifier to interact with the server, possibly imbuing the session identifier with the user's credentials or confidential information. Third, the attacker uses the session identifier to interact with server directly, possibly obtaining the user's authority or confidential information.

#### 8.5. Weak Confidentiality

Cookies do not provide isolation by port. If a cookie is readable by a service running on one port, the cookie is also readable by a service running on another port of the same server. If a cookie is writable by a service on one port, the cookie is also writable by a service running on another port of the same server. For this reason, servers SHOULD NOT both run mutually distrusting services on different ports of the same host and use cookies to store security-sensitive information.

Cookies do not provide isolation by scheme. Although most commonly used with the http and https schemes, the cookies for a given host might also be available to other schemes, such as ftp and gopher. Although this lack of isolation by scheme is most apparent in non-HTTP APIs that permit access to cookies (e.g., HTML's document.cookie API), the lack of isolation by scheme is actually present in requirements for processing cookies themselves (e.g., consider retrieving a URI with the gopher scheme via HTTP).

Cookies do not always provide isolation by path. Although the network-level protocol does not send cookies stored for one path to another, some user agents expose cookies via non-HTTP APIs, such as HTML's document.cookie API. Because some of these user agents (e.g., web browsers) do not isolate resources received from different paths, a resource retrieved from one path might be able to access cookies stored for another path.

## 8.6. Weak Integrity

Cookies do not provide integrity guarantees for sibling domains (and their subdomains). For example, consider `foo.site.example` and `bar.site.example`. The `foo.site.example` server can set a cookie with a Domain attribute of `"site.example"` (possibly overwriting an existing `"site.example"` cookie set by `bar.site.example`), and the user agent will include that cookie in HTTP requests to `bar.site.example`. In the worst case, `bar.site.example` will be unable to distinguish this cookie from a cookie it set itself. The `foo.site.example` server might be able to leverage this ability to mount an attack against `bar.site.example`.

Even though the Set-Cookie header field supports the Path attribute, the Path attribute does not provide any integrity protection because the user agent will accept an arbitrary Path attribute in a Set-Cookie header field. For example, an HTTP response to a request for `http://site.example/foo/bar` can set a cookie with a Path attribute of `"/qux"`. Consequently, servers SHOULD NOT both run mutually distrusting services on different paths of the same host and use cookies to store security-sensitive information.

An active network attacker can also inject cookies into the Cookie header field sent to `https://site.example/` by impersonating a response from `http://site.example/` and injecting a Set-Cookie header field. The HTTPS server at `site.example` will be unable to distinguish these cookies from cookies that it set itself in an HTTPS response. An active network attacker might be able to leverage this ability to mount an attack against `site.example` even if `site.example` uses HTTPS exclusively.

Servers can partially mitigate these attacks by encrypting and signing the contents of their cookies, or by naming the cookie with the `__Secure-` prefix. However, using cryptography does not mitigate the issue completely because an attacker can replay a cookie he or she received from the authentic `site.example` server in the user's session, with unpredictable results.

Finally, an attacker might be able to force the user agent to delete cookies by storing a large number of cookies. Once the user agent reaches its storage limit, the user agent will be forced to evict some cookies. Servers SHOULD NOT rely upon user agents retaining cookies.

### 8.7. Reliance on DNS

Cookies rely upon the Domain Name System (DNS) for security. If the DNS is partially or fully compromised, the cookie protocol might fail to provide the security properties required by applications.

### 8.8. SameSite Cookies

#### 8.8.1. Defense in depth

"SameSite" cookies offer a robust defense against CSRF attack when deployed in strict mode, and when supported by the client. It is, however, prudent to ensure that this designation is not the extent of a site's defense against CSRF, as same-site navigations and submissions can certainly be executed in conjunction with other attack vectors such as cross-site scripting.

Developers are strongly encouraged to deploy the usual server-side defenses (CSRF tokens, ensuring that "safe" HTTP methods are idempotent, etc) to mitigate the risk more fully.

Additionally, client-side techniques such as those described in [app-isolation] may also prove effective against CSRF, and are certainly worth exploring in combination with "SameSite" cookies.

#### 8.8.2. Top-level Navigations

Setting the SameSite attribute in "strict" mode provides robust defense in depth against CSRF attacks, but has the potential to confuse users unless sites' developers carefully ensure that their cookie-based session management systems deal reasonably well with top-level navigations.

Consider the scenario in which a user reads their email at MegaCorp Inc's webmail provider <https://site.example/>. They might expect that clicking on an emailed link to <https://projects.example/secret/> project would show them the secret project that they're authorized to see, but if [https://projects.example](https://projects.example/) has marked their session cookies as SameSite=Strict, then this cross-site navigation won't send them along with the request. <https://projects.example> will render a 404 error to avoid leaking secret information, and the user will be quite confused.

Developers can avoid this confusion by adopting a session management system that relies on not one, but two cookies: one conceptually granting "read" access, another granting "write" access. The latter could be marked as SameSite=Strict, and its absence would prompt a reauthentication step before executing any non-idempotent action.

The former could be marked as `SameSite=Lax`, in order to allow users access to data via top-level navigation, or `SameSite=None`, to permit access in all contexts (including cross-site embedded contexts).

#### 8.8.3. Mashups and Widgets

The `Lax` and `Strict` values for the `SameSite` attribute are inappropriate for some important use-cases. In particular, note that content intended for embedding in cross-site contexts (social networking widgets or commenting services, for instance) will not have access to same-site cookies. Cookies which are required in these situations should be marked with `SameSite=None` to allow access in cross-site contexts.

Likewise, some forms of Single-Sign-On might require cookie-based authentication in a cross-site context; these mechanisms will not function as intended with same-site cookies and will also require `SameSite=None`.

#### 8.8.4. Server-controlled

`SameSite` cookies in and of themselves don't do anything to address the general privacy concerns outlined in Section 7.1 of [RFC6265]. The "`SameSite`" attribute is set by the server, and serves to mitigate the risk of certain kinds of attacks that the server is worried about. The user is not involved in this decision. Moreover, a number of side-channels exist which could allow a server to link distinct requests even in the absence of cookies (for example, connection and/or socket pooling between same-site and cross-site requests).

#### 8.8.5. Reload navigations

Requests issued for reloads triggered through user interface elements (such as a refresh button on a toolbar) are same-site only if the reloaded document was originally navigated to via a same-site request. This differs from the handling of other reload navigations, which are always same-site if top-level, since the source browsing context's active document is precisely the document being reloaded.

This special handling of reloads triggered through a user interface element avoids sending SameSite cookies on user-initiated reloads if they were withheld on the original navigation (i.e., if the initial navigation were cross-site). If the reload navigation were instead considered same-site, and sent all the initially withheld SameSite cookies, the security benefits of withholding the cookies in the first place would be nullified. This is especially important given that the absence of SameSite cookies withheld on a cross-site navigation request may lead to visible site breakage, prompting the user to trigger a reload.

For example, suppose the user clicks on a link from `https://attacker.example/` to `https://victim.example/`. This is a cross-site request, so `SameSite=Strict` cookies are withheld. Suppose this causes `https://victim.example/` to appear broken, because the site only displays its sensitive content if a particular SameSite cookie is present in the request. The user, frustrated by the unexpectedly broken site, presses refresh on their browser's toolbar. To now consider the reload request same-site and send the initially withheld SameSite cookie would defeat the purpose of withholding it in the first place, as the reload navigation triggered through the user interface may replay the original (potentially malicious) request. Thus, the reload request should be considered cross-site, like the request that initially navigated to the page.

#### 8.8.6. Top-level requests with "unsafe" methods

The "Lax" enforcement mode described in Section 5.4.7.1 allows a cookie to be sent with a cross-site HTTP request if and only if it is a top-level navigation with a "safe" HTTP method. Implementation experience shows that this is difficult to apply as the default behavior, as some sites may rely on cookies not explicitly specifying a SameSite attribute being included on top-level cross-site requests with "unsafe" HTTP methods (as was the case prior to the introduction of the SameSite attribute).

For example, a login flow may involve a cross-site top-level POST request to an endpoint which expects a cookie with login information. For such a cookie, "Lax" enforcement is not appropriate, as it would cause the cookie to be excluded due to the unsafe HTTP request method. On the other hand, "None" enforcement would allow the cookie to be sent with all cross-site requests, which may not be desirable due to the cookie's sensitive contents.

The "Lax-allowing-unsafe" enforcement mode described in Section 5.4.7.2 retains some of the protections of "Lax" enforcement (as compared to "None") while still allowing cookies to be sent cross-site with unsafe top-level requests.



As a more permissive variant of "Lax" mode, "Lax-allowing-unsafe" mode necessarily provides fewer protections against CSRF. Ultimately, the provision of such an enforcement mode should be seen as a temporary, transitional measure to ease adoption of "Lax" enforcement by default.

## 9. IANA Considerations

### 9.1. Cookie

The permanent message header field registry (see [RFC3864]) needs to be updated with the following registration:

Header field name: Cookie

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document: this specification (Section 5.6.1)

### 9.2. Set-Cookie

The permanent message header field registry (see [RFC3864]) needs to be updated with the following registration:

Header field name: Set-Cookie

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document: this specification (Section 5.4)

### 9.3. Cookie Attribute Registry

IANA is requested to create the "Cookie Attribute Registry", defining the name space of attribute used to control cookies' behavior. The registry should be maintained at <https://www.iana.org/assignments/cookie-attribute-names> (<https://www.iana.org/assignments/cookie-attribute-names>).

### 9.3.1. Procedure

Each registered attribute name is associated with a description, and a reference detailing how the attribute is to be processed and stored.

New registrations happen on a "RFC Required" basis (see Section 4.7 of [RFC8126]). The attribute to be registered MUST match the extension-av syntax defined in Section 4.1.1. Note that attribute names are generally defined in CamelCase, but technically accepted case-insensitively.

### 9.3.2. Registration

The "Cookie Attribute Registry" should be created with the registrations below:

Name	Reference
Domain	Section 4.1.2.3 of this document
Expires	Section 4.1.2.1 of this document
HttpOnly	Section 4.1.2.6 of this document
Max-Age	Section 4.1.2.2 of this document
Path	Section 4.1.2.4 of this document
SameSite	Section 4.1.2.7 of this document
Secure	Section 4.1.2.5 of this document

Table 1

## 10. References

### 10.1. Normative References

- [DOM-DOCUMENT-COOKIE] WHATWG, "HTML - Living Standard", 18 May 2021, <<https://html.spec.whatwg.org/#dom-document-cookie>>.
- [FETCH] van Kesteren, A., "Fetch", n.d., <<https://fetch.spec.whatwg.org/>>.

- [HTML] Hickson, I., Pieters, S., van Kesteren, A., Jägenstedt, P., and D. Denicola, "HTML", n.d., <<https://html.spec.whatwg.org/>>.
- [HTTPSEM] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>>.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/rfc/rfc1034>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<https://www.rfc-editor.org/rfc/rfc1123>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC3490] Costello, A., "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003, <<https://www.rfc-editor.org/rfc/rfc3490>>. See Section 6.3 for an explanation why the normative reference to an obsoleted specification is needed.
- [RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <<https://www.rfc-editor.org/rfc/rfc4790>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/rfc/rfc5234>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/rfc/rfc5890>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/rfc/rfc6454>>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.

[SAMESITE] WHATWG, "HTML - Living Standard", 26 January 2021, <<https://html.spec.whatwg.org/#same-site>>.

[SERVICE-WORKERS] Russell, A., Song, J., and J. Archibald, "Service Workers", n.d., <<http://www.w3.org/TR/service-workers/>>.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

## 10.2. Informative References

[Aggarwal2010] Aggarwal, G., Burzstein, E., Jackson, C., and D. Boneh, "An Analysis of Private Browsing Modes in Modern Browsers", 2010, <[http://www.usenix.org/events/sec10/tech/full\\_papers/Aggarwal.pdf](http://www.usenix.org/events/sec10/tech/full_papers/Aggarwal.pdf)>.

[app-isolation] Chen, E., Bau, J., Reis, C., Barth, A., and C. Jackson, "App Isolation - Get the Security of Multiple Browsers with Just One", 2011, <<http://www.collinjakson.com/research/papers/appisolation.pdf>>.

[CSRF] Barth, A., Jackson, C., and J. Mitchell, "Robust Defenses for Cross-Site Request Forgery", DOI 10.1145/1455770.1455782, ISBN 978-1-59593-810-7, ACM CCS '08: Proceedings of the 15th ACM conference on Computer and communications security (pages 75-88), October 2008, <<http://portal.acm.org/citation.cfm?id=1455770.1455782>>.

[I-D.ietf-httpbis-cookie-alone] West, M., "Deprecate modification of 'secure' cookies from non-secure origins", Work in Progress, Internet-Draft, draft-ietf-httpbis-cookie-alone-01, 5 September 2016, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cookie-alone-01>>.

- [I-D.ietf-httpbis-cookie-prefixes]  
West, M., "Cookie Prefixes", Work in Progress, Internet-Draft, draft-ietf-httpbis-cookie-prefixes-00, 23 February 2016, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cookie-prefixes-00>>.
- [I-D.ietf-httpbis-cookie-same-site]  
West, M. and M. Goodwin, "Same-Site Cookies", Work in Progress, Internet-Draft, draft-ietf-httpbis-cookie-same-site-00, 20 June 2016, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cookie-same-site-00>>.
- [prerendering]  
Bentzel, C., "Chrome Prerendering", n.d., <<https://www.chromium.org/developers/design-documents/prerender>>.
- [PSL] "Public Suffix List", n.d., <<https://publicsuffix.org/list/>>.
- [RFC2109] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2109, DOI 10.17487/RFC2109, February 1997, <<https://www.rfc-editor.org/rfc/rfc2109>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/rfc/rfc2818>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/rfc/rfc3629>>.
- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <<https://www.rfc-editor.org/rfc/rfc3864>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.

- [RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, DOI 10.17487/RFC5895, September 2010, <<https://www.rfc-editor.org/rfc/rfc5895>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/rfc/rfc6265>>.
- [RFC7034] Ross, D. and T. Gondrom, "HTTP Header Field X-Frame-Options", RFC 7034, DOI 10.17487/RFC7034, October 2013, <<https://www.rfc-editor.org/rfc/rfc7034>>.
- [UTS46] Davis, M. and M. Suignard, "Unicode IDNA Compatibility Processing", UNICODE Unicode Technical Standards # 46, June 2016, <<http://unicode.org/reports/tr46/>>.

## Appendix A. Changes

### A.1. draft-ietf-httpbis-rfc6265bis-00

- \* Port [RFC6265] to Markdown. No (intentional) normative changes.

### A.2. draft-ietf-httpbis-rfc6265bis-01

- \* Fixes to formatting caused by mistakes in the initial port to Markdown:
  - <https://github.com/httpwg/http-extensions/issues/243>  
(<https://github.com/httpwg/http-extensions/issues/243>)
  - <https://github.com/httpwg/http-extensions/issues/246>  
(<https://github.com/httpwg/http-extensions/issues/246>)
- \* Addresses errata 3444 by updating the path-value and extension-av grammar, errata 4148 by updating the day-of-month, year, and time grammar, and errata 3663 by adding the requested note.  
[https://www.rfc-editor.org/errata\\_search.php?rfc=6265](https://www.rfc-editor.org/errata_search.php?rfc=6265)  
([https://www.rfc-editor.org/errata\\_search.php?rfc=6265](https://www.rfc-editor.org/errata_search.php?rfc=6265))
- \* Dropped Cookie2 and Set-Cookie2 from the IANA Considerations section: <https://github.com/httpwg/http-extensions/issues/247>  
(<https://github.com/httpwg/http-extensions/issues/247>)
- \* Merged the recommendations from [I-D.ietf-httpbis-cookie-alone], removing the ability for a non-secure origin to set cookies with a 'secure' flag, and to overwrite cookies whose 'secure' flag is true.

- \* Merged the recommendations from [I-D.ietf-httpbis-cookie-prefixes], adding `__Secure-` and `__Host-` cookie name prefix processing instructions.

#### A.3. draft-ietf-httpbis-rfc6265bis-02

- \* Merged the recommendations from [I-D.ietf-httpbis-cookie-same-site], adding support for the `SameSite` attribute.
- \* Closed a number of editorial bugs:
  - Clarified address bar behavior for `SameSite` cookies:  
<https://github.com/httpwg/http-extensions/issues/201>  
(<https://github.com/httpwg/http-extensions/issues/201>)
  - Added the word "Cookies" to the document's name:  
<https://github.com/httpwg/http-extensions/issues/204>  
(<https://github.com/httpwg/http-extensions/issues/204>)
  - Clarified that the `__Host-` prefix requires an explicit `Path` attribute: <https://github.com/httpwg/http-extensions/issues/222>  
(<https://github.com/httpwg/http-extensions/issues/222>)
  - Expanded the options for dealing with third-party cookies to include a brief mention of partitioning based on first-party:  
<https://github.com/httpwg/http-extensions/issues/248>  
(<https://github.com/httpwg/http-extensions/issues/248>)
  - Noted that double-quotes in cookie values are part of the value, and are not stripped: <https://github.com/httpwg/http-extensions/issues/295> (<https://github.com/httpwg/http-extensions/issues/295>)
  - Fixed the "site for cookies" algorithm to return something that makes sense: <https://github.com/httpwg/http-extensions/issues/302> (<https://github.com/httpwg/http-extensions/issues/302>)

#### A.4. draft-ietf-httpbis-rfc6265bis-03

- \* Clarified handling of invalid `SameSite` values:  
<https://github.com/httpwg/http-extensions/issues/389>  
(<https://github.com/httpwg/http-extensions/issues/389>)

- \* Reflect widespread implementation practice of including a cookie's host-only-flag when calculating its uniqueness:  
<https://github.com/httpwg/http-extensions/issues/199>  
(<https://github.com/httpwg/http-extensions/issues/199>)
- \* Introduced an explicit "None" value for the SameSite attribute:  
<https://github.com/httpwg/http-extensions/issues/788>  
(<https://github.com/httpwg/http-extensions/issues/788>)

#### A.5. draft-ietf-httpbis-rfc6265bis-04

- \* Allow SameSite cookies to be set for all top-level navigations.  
<https://github.com/httpwg/http-extensions/issues/594>  
(<https://github.com/httpwg/http-extensions/issues/594>)
- \* Treat Set-Cookie: token as creating the cookie ("", "token"):  
<https://github.com/httpwg/http-extensions/issues/159>  
(<https://github.com/httpwg/http-extensions/issues/159>)
- \* Reject cookies with neither name nor value (e.g. Set-Cookie: = and Set-Cookie:: <https://github.com/httpwg/http-extensions/issues/159> (<https://github.com/httpwg/http-extensions/issues/159>))
- \* Clarified behavior of multiple SameSite attributes in a cookie string: <https://github.com/httpwg/http-extensions/issues/901>  
(<https://github.com/httpwg/http-extensions/issues/901>)

#### A.6. draft-ietf-httpbis-rfc6265bis-05

- \* Typos and editorial fixes: <https://github.com/httpwg/http-extensions/pull/1035> (<https://github.com/httpwg/http-extensions/pull/1035>), <https://github.com/httpwg/http-extensions/pull/1038> (<https://github.com/httpwg/http-extensions/pull/1038>), <https://github.com/httpwg/http-extensions/pull/1040> (<https://github.com/httpwg/http-extensions/pull/1040>), <https://github.com/httpwg/http-extensions/pull/1047> (<https://github.com/httpwg/http-extensions/pull/1047>).

#### A.7. draft-ietf-httpbis-rfc6265bis-06

- \* Editorial fixes: <https://github.com/httpwg/http-extensions/issues/1059> (<https://github.com/httpwg/http-extensions/issues/1059>), <https://github.com/httpwg/http-extensions/issues/1158> (<https://github.com/httpwg/http-extensions/issues/1158>).



- \* Created a registry for cookie attribute names:  
<https://github.com/httpwg/http-extensions/pull/1060>  
(<https://github.com/httpwg/http-extensions/pull/1060>).
- \* Tweaks to ABNF for cookie-pair and the Cookie header production:  
<https://github.com/httpwg/http-extensions/issues/1074>  
(<https://github.com/httpwg/http-extensions/issues/1074>),  
<https://github.com/httpwg/http-extensions/issues/1119>  
(<https://github.com/httpwg/http-extensions/issues/1119>).
- \* Fixed serialization for nameless/valueless cookies:  
<https://github.com/httpwg/http-extensions/pull/1143>  
(<https://github.com/httpwg/http-extensions/pull/1143>).
- \* Converted a normative reference to Mozilla's Public Suffix List [PSL] into an informative reference: <https://github.com/httpwg/http-extensions/issues/1159> (<https://github.com/httpwg/http-extensions/issues/1159>).

#### A.8. draft-ietf-httpbis-rfc6265bis-07

- \* Moved instruction to ignore cookies with empty cookie-name and cookie-value from Section 5.4 to Section 5.5 to ensure that they apply to cookies created without parsing a cookie string:  
<https://github.com/httpwg/http-extensions/issues/1234>  
(<https://github.com/httpwg/http-extensions/issues/1234>).
- \* Add a default enforcement value to the same-site-flag, equivalent to "SameSite=Lax": <https://github.com/httpwg/http-extensions/pull/1325> (<https://github.com/httpwg/http-extensions/pull/1325>).
- \* Require a Secure attribute for "SameSite=None":  
<https://github.com/httpwg/http-extensions/pull/1323>  
(<https://github.com/httpwg/http-extensions/pull/1323>).
- \* Consider scheme when running the same-site algorithm:  
<https://github.com/httpwg/http-extensions/pull/1324>  
(<https://github.com/httpwg/http-extensions/pull/1324>).

#### A.9. draft-ietf-httpbis-rfc6265bis-08

- \* Define "same-site" for reload navigation requests, e.g. those triggered via user interface elements: <https://github.com/httpwg/http-extensions/pull/1384> (<https://github.com/httpwg/http-extensions/pull/1384>)

- \* Consider redirects when defining same-site:  
<https://github.com/httpwg/http-extensions/pull/1348>  
(<https://github.com/httpwg/http-extensions/pull/1348>)
- \* Align on using HTML terminology for origins:  
<https://github.com/httpwg/http-extensions/pull/1416>  
(<https://github.com/httpwg/http-extensions/pull/1416>)
- \* Modify cookie parsing and creation algorithms in Section 5.4 and Section 5.5 to explicitly handle control characters:  
<https://github.com/httpwg/http-extensions/pull/1420>  
(<https://github.com/httpwg/http-extensions/pull/1420>)
- \* Refactor cookie retrieval algorithm to support non-HTTP APIs:  
<https://github.com/httpwg/http-extensions/pull/1428>  
(<https://github.com/httpwg/http-extensions/pull/1428>)
- \* Define "Lax-allowing-unsafe" SameSite enforcement mode:  
<https://github.com/httpwg/http-extensions/pull/1435>  
(<https://github.com/httpwg/http-extensions/pull/1435>)
- \* Consistently use "header field" (vs 'header'):  
<https://github.com/httpwg/http-extensions/pull/1527>  
(<https://github.com/httpwg/http-extensions/pull/1527>)

#### A.10. draft-ietf-httpbis-rfc6265bis-09

- \* Update cookie size requirements: <https://github.com/httpwg/http-extensions/pull/1563> (<https://github.com/httpwg/http-extensions/pull/1563>)
- \* Reject cookies with control characters: <https://github.com/httpwg/http-extensions/pull/1576> (<https://github.com/httpwg/http-extensions/pull/1576>)
- \* No longer treat horizontal tab as a control character:  
<https://github.com/httpwg/http-extensions/pull/1589>  
(<https://github.com/httpwg/http-extensions/pull/1589>)
- \* Specify empty domain attribute handling:  
<https://github.com/httpwg/http-extensions/pull/1709>  
(<https://github.com/httpwg/http-extensions/pull/1709>)

#### A.11. draft-ietf-httpbis-rfc6265bis-10

- \* Standardize Max-Age/Expires upper bound:  
<https://github.com/httpwg/http-extensions/pull/1732>  
(<https://github.com/httpwg/http-extensions/pull/1732>)

## Acknowledgements

RFC 6265 was written by Adam Barth. This document is an update of RFC 6265, adding features and aligning the specification with the reality of today's deployments. Here, we're standing upon the shoulders of a giant since the majority of the text is still Adam's.

## Authors' Addresses

Lily Chen (editor)  
Google LLC  
Email: [chlily@google.com](mailto:chlily@google.com)

Steven Englehardt (editor)  
Mozilla  
Email: [senglehardt@mozilla.com](mailto:senglehardt@mozilla.com)

Mike West (editor)  
Google LLC  
Email: [mkwst@google.com](mailto:mkwst@google.com)  
URI: <https://mikewest.org/>

John Wilander (editor)  
Apple, Inc  
Email: [wilander@apple.com](mailto:wilander@apple.com)

HTTP  
Internet-Draft  
Updates: 7234 (if approved)  
Intended status: Standards Track  
Expires: May 5, 2020

M. Nottingham  
Fastly  
November 2, 2019

HTTP Representation Variants  
draft-ietf-httpbis-variants-06

Abstract

This specification introduces an alternative way to select a HTTP response from a cache based upon its request headers, using the HTTP "Variants" and "Variant-Key" response header fields. Its aim is to make HTTP proactive content negotiation more cache-friendly.

Note to Readers

\_RFC EDITOR: please remove this section before publication\_

Discussion of this draft takes place on the HTTP working group mailing list ([ietf-http-wg@w3.org](mailto:ietf-http-wg@w3.org)), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/> [1].

Working Group information can be found at <https://httpwg.github.io/> [2]; source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions/labels/variants> [3].

There is a prototype implementation of the algorithms herein at <https://github.com/mnot/variants-toy> [4].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 5, 2020.

## Copyright Notice

Copyright (c) 2019 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 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 . . . . .	3
1.1. Notational Conventions . . . . .	5
2. The "Variants" HTTP Header Field . . . . .	5
2.1. Relationship to Vary . . . . .	7
3. The "Variant-Key" HTTP Header Field . . . . .	7
4. Cache Behaviour . . . . .	9
4.1. Compute Possible Keys . . . . .	10
4.2. Check Vary . . . . .	11
4.3. Example of Cache Behaviour . . . . .	11
4.3.1. A Variant Missing From the Cache . . . . .	12
4.3.2. Variants That Don't Overlap the Client's Request . . . . .	13
5. Origin Server Behaviour . . . . .	13
5.1. Examples . . . . .	14
5.1.1. Single Variant . . . . .	14
5.1.2. Multiple Variants . . . . .	15
5.1.3. Partial Coverage . . . . .	15
6. Defining Content Negotiation Using Variants . . . . .	16
7. IANA Considerations . . . . .	16
8. Security Considerations . . . . .	17
9. References . . . . .	17
9.1. Normative References . . . . .	17
9.2. Informative References . . . . .	18
9.3. URIs . . . . .	18
Appendix A. Variants for Existing Content Negotiation Mechanisms . . . . .	19
A.1. Accept . . . . .	19
A.2. Accept-Encoding . . . . .	20
A.3. Accept-Language . . . . .	20
A.4. Cookie . . . . .	21
Acknowledgements . . . . .	22
Author's Address . . . . .	23

## 1. Introduction

HTTP proactive content negotiation ([RFC7231], Section 3.4.1) is seeing renewed interest, both for existing request headers like Accept-Language and for newer ones (for example, see [I-D.ietf-httpbis-client-hints]).

Successfully reusing negotiated responses that have been stored in a HTTP cache requires establishment of a secondary cache key ([RFC7234], Section 4.1). Currently, the Vary header ([RFC7231], Section 7.1.4) does this by nominating a set of request headers. Their values collectively form the secondary cache key for a given response.

HTTP's caching model allows a certain amount of latitude in normalising those request header field values, so as to increase the chances of a cache hit while still respecting the semantics of that header. However, normalisation is not formally defined, leading to infrequent implementation in cache, and divergence of behaviours when it is.

Even when the headers' semantics are understood, a cache does not know enough about the possible alternative representations available on the origin server to make an appropriate decision.

For example, if a cache has stored the following request/response pair:

```
GET /foo HTTP/1.1
Host: www.example.com
Accept-Language: en;q=0.5, fr;q=1.0
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Language: en
Vary: Accept-Language
Transfer-Encoding: chunked
```

[English content]

Provided that the cache has full knowledge of the semantics of Accept-Language and Content-Language, it will know that an English representation is available and might be able to infer that a French representation is not available. But, it does not know (for example) whether a Japanese representation is available without making another request, incurring possibly unnecessary latency.

This specification introduces the HTTP Variants response header field (Section 2) to enumerate the available variant representations on the origin server, to provide clients and caches with enough information to properly satisfy requests – either by selecting a response from cache or by forwarding the request towards the origin – by following the algorithm defined in Section 4.

Its companion Variant-Key response header field (Section 3) indicates the applicable key(s) that the response is associated with, so that it can be reliably reused in the future. Effectively, it allows the specification of a request header field to define how it affects the secondary cache key.

When this specification is in use, the example above might become:

```
GET /foo HTTP/1.1
Host: www.example.com
Accept-Language: en;q=0.5, fr;q=1.0
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Language: en
Vary: Accept-Language
Variants: Accept-Language;de;en;jp
Variant-Key: en
Transfer-Encoding: chunked
```

[English content]

Proactive content negotiation mechanisms that wish to be used with Variants need to define how to do so explicitly; see Section 6. As a result, it is best suited for negotiation over request headers that are well-understood.

Variants also works best when content negotiation takes place over a constrained set of representations; since each variant needs to be listed in the header field, it is ill-suited for open-ended sets of representations.

Variants can be seen as a simpler version of the Alternates header field introduced by [RFC2295]; unlike that mechanism, Variants does not require specification of each combination of attributes, and does not assume that each combination has a unique URL.

### 1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] but relies on Structured Headers from [I-D.ietf-httpbis-header-structure] for parsing.

Additionally, it uses the "field-name" rule from [RFC7230], "type", "subtype", "content-coding" and "language-range" from [RFC7231], and "cookie-name" from [RFC6265].

### 2. The "Variants" HTTP Header Field

The Variants HTTP response header field indicates what representations are available for a given resource at the time that the response is produced, by enumerating the request header fields that it varies on, along with a representation of the values that are available for each.

Variants is a Structured Header Dictionary (Section 3.2 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

```
Variants      = sh-dict
```

Each member-name represents the field-name of a request header that is part of the secondary cache key; each member-value is an inner-list of strings or tokens that convey representations of potential values for that header field, hereafter referred to as "available-values".

If Structured Header parsing fails or a member's value does not have the structure outlined above, the client MUST treat the representation as having no Variants header field.

Note that an available-value that is a token is interpreted as a string containing the same characters, and vice versa.

So, given this example header field:

```
Variants: Accept-Encoding=(gzip)
```



a recipient can infer that the only content-coding available for that resource is "gzip" (along with the "identity" non-encoding; see Appendix A.2).

Given:

Variants: accept-encoding=()

a recipient can infer that no content-codings (beyond identity) are supported. Note that as always, field-name is case-insensitive.

A more complex example:

Variants: Accept-Encoding=(gzip br), Accept-Language=(en fr)

Here, recipients can infer that two content-codings in addition to "identity" are available, as well as two content languages. Note that, as with all Structured Header dictionaries, they might occur in the same header field or separately, like this:

Variants: Accept-Encoding=(gzip brotli)

Variants: Accept-Language=(en fr)

The ordering of available-values is significant, as it might be used by the header's algorithm for selecting a response (in this example, the first language is the default; see Appendix A.3).

The ordering of the request header fields themselves indicates descending application of preferences; in the example above, a cache that has all of the possible permutations stored will honour the client's preferences for Accept-Encoding before honouring Accept-Language.

Origin servers SHOULD consistently send Variant header fields on all cacheable (as per [RFC7234], Section 3) responses for a resource, since its absence will trigger caches to fall back to Vary processing.

Likewise, servers MUST send the Variant-Key response header field when sending Variants, since its absence means that the stored response will not be reused when this specification is implemented.

\_RFC EDITOR: Please remove the next paragraph before publication.\_

Implementations of drafts of this specification MUST implement an HTTP header field named "Variants-##" instead of the "Variants" header field specified by the final RFC, with "##" replaced by the

draft number being implemented. For example, implementations of draft-ietf-httpbis-variants-05 would implement "Variants-05".

### 2.1. Relationship to Vary

This specification updates [RFC7234] to allow caches that implement it to ignore request header fields in the Vary header for the purposes of secondary cache key calculation ([RFC7234], Section 4.1) when their semantics are implemented as per this specification and their corresponding response header field is listed in Variants.

If any member of the Vary header does not have a corresponding variant that is understood by the implementation, it is still subject to the requirements there.

See Section 5.1.3 for an example.

In practice, implementation of Vary varies considerably. As a result, cache efficiency might drop considerably when Variants does not contain all of the headers referenced by Vary, because some implementations might choose to disable Variants processing when this is the case.

### 3. The "Variant-Key" HTTP Header Field

The Variant-Key HTTP response header field identifies one or more sets of available-values that identify the secondary cache key(s) that the response it occurs within are associated with.

Variant-Key is a Structured Header List (Section 3.1 of [I-D.ietf-httpbis-header-structure]) whose members are inner-lists of strings or tokens. Its ABNF is:

```
Variant-Key      = sh-list
```

Each member MUST be an inner-list, and MUST itself have the same number of members as there are members of the representation's Variants header field. If not, the client MUST treat the representation as having no Variant-Key header field.

Each member identifies a list of available-values corresponding to the header field-names in the Variants header field, thereby identifying a secondary cache key that can be used with this response. These available-values do not need to explicitly appear in the Variants header field; they can be interpreted by the algorithm specific to processing that field. For example, Accept-Encoding defines an implicit "identity" available-value (Appendix A.2).

Each inner-list member is treated as identifying an available-value for the corresponding variant-axis' field-name. Any list-member that is a token is interpreted as a string containing the same characters.

For example:

```
Variants: Accept-Encoding=(gzip br), Accept-Language=(en fr)
Variant-Key: (gzip fr)
```

This header pair indicates that the representation has a "gzip" content-coding and "fr" content-language.

If the response can be used to satisfy more than one request, they can be listed in additional members. For example:

```
Variants: Accept-Encoding=(gzip br), Accept-Language=(en fr)
Variant-Key: (gzip fr), ("identity" fr)
```

indicates that this response can be used for requests whose Accept-Encoding algorithm selects "gzip" or "identity", as long as the Accept-Language algorithm selects "fr" - perhaps because there is no gzip-compressed French representation.

When more than one Variant-Key value is in a response, the first one present MUST correspond to the request that caused that response to be generated. For example:

```
Variants: Accept-Encoding=(gzip br), Accept-Language=(en fr)
Variant-Key: (gzip fr), (identity fr), (br fr oops)
```

is treated as if the Variant-Key header were completely absent, which will tend to disable caching for the representation that contains it.

Note that in

```
Variant-Key: (gzip fr)
Variant-Key: ("gzip " fr)
```

The whitespace after "gzip" in the first header field value is excluded by the parsing algorithm, but the whitespace in the second header field value is included by the string parsing algorithm. This will likely cause the second header field value to fail to match client requests.

\_RFC EDITOR: Please remove the next paragraph before publication.\_

Implementations of drafts of this specification MUST implement an HTTP header field named "Variant-Key-##" instead of the "Variant-Key"

header field specified by the final RFC, with "##" replaced by the draft number being implemented. For example, implementations of draft-ietf-httpbis-variants-05 would implement "Variant-Key-05".

#### 4. Cache Behaviour

Caches that implement the Variants header field and the relevant semantics of the field-names it contains can use that knowledge to either select an appropriate stored representation, or forward the request if no appropriate representation is stored.

They do so by running this algorithm (or its functional equivalent) upon receiving a request:

Given incoming-request (a mapping of field-names to field-values, after being combined as allowed by Section 3.2.2 of [RFC7230]), and stored-responses (a list of stored responses suitable for reuse as defined in Section 4 of [RFC7234], excepting the requirement to calculate a secondary cache key):

1. If stored-responses is empty, return an empty list.
2. Order stored-responses by the "Date" header field, most recent to least recent.
3. Let sorted-variants be an empty list.
4. If the freshest member of stored-responses (as per [RFC7234], Section 4.2) has one or more "Variants" header field(s) that successfully parse according to Section 2:
  1. Select one member of stored-responses with a "Variants" header field-value(s) that successfully parses according to Section 2 and let variants-header be this parsed value. This SHOULD be the most recent response, but MAY be from an older one as long as it is still fresh.
  2. For each variant-axis in variants-header:
    1. If variant-axis' field-name corresponds to the request header field identified by a content negotiation mechanism that the implementation supports:
      1. Let request-value be the field-value associated with field-name in incoming-request, or null if field-name is not in incoming-request.

2. Let sorted-values be the result of running the algorithm defined by the content negotiation mechanism with request-value and variant-axis' available-values.
3. Append sorted-values to sorted-variants.

At this point, sorted-variants will be a list of lists, each member of the top-level list corresponding to a variant-axis in the Variants header field-value, containing zero or more items indicating available-values that are acceptable to the client, in order of preference, greatest to least.

5. Return result of running Compute Possible Keys (Section 4.1) on sorted-variants, an empty list and an empty list.

This returns a list of lists of strings suitable for comparing to the parsed Variant-Keys (Section 3) that represent possible responses on the server that can be used to satisfy the request, in preference order, provided that their secondary cache key (after removing the headers covered by Variants) matches. Section 4.2 illustrates one way to do this.

#### 4.1. Compute Possible Keys

This algorithm computes the cross-product of the elements of key-facets.

Given key-facets (a list of lists of strings), and key-stub (a list of strings representing a partial key), and possible-keys (a list of lists of strings):

1. Let values be the first member of key-facets.
2. Let remaining-facets be a copy of all of the members of key-facets except the first.
3. For each value in values:
  1. Let this-key be a copy of key-stub.
  2. Append value to this-key.
  3. If remaining-facets is empty, append this-key to possible-keys.
  4. Otherwise, run Compute Possible Keys on remaining-facets, this-key and possible-keys.

4. Return possible-keys.

#### 4.2. Check Vary

This algorithm is an example of how an implementation can meet the requirement to apply the members of the Vary header field that are not covered by Variants.

Given incoming-request (a mapping of field-names to field-values, after being combined as allowed by Section 3.2.2 of [RFC7230]), and stored-response (a stored response):

1. Let filtered-vary be the field-value(s) of stored-response's "Vary" header field.
2. Let processed-variants be a list containing the request header fields that identify the content negotiation mechanisms supported by the implementation.
3. Remove any member of filtered-vary that is a case-insensitive match for a member of processed-variants.
4. If the secondary cache key (as calculated in [RFC7234], Section 4.1) for stored\_response matches incoming-request, using filtered-vary for the value of the "Vary" response header, return True.
5. Return False.

This returns a Boolean that indicates whether stored-response can be used to satisfy the request.

Note that implementation of the Vary header field varies in practice, and the algorithm above illustrates only one way to apply it. It is equally viable to forward the request if there is a request header listed in Vary but not Variants.

#### 4.3. Example of Cache Behaviour

For example, if the selected variants-header was:

Variants: Accept-Language=(en fr de), Accept-Encoding=(gzip br)

and the request contained the headers:

Accept-Language: fr;q=1.0, en;q=0.1  
Accept-Encoding: gzip

Then the sorted-variants would be:

```
[
  ["fr", "en"]           // prefers French, will accept English
  ["gzip", "identity"] // prefers gzip encoding, will accept identity
]
```

Which means that the result of the Cache Behaviour algorithm would be:

```
[
  ["fr", "gzip"],
  ["fr", "identity"],
  ["en", "gzip"],
  ["en", "identity"]
]
```

Representing a first preference of a French, gzip'd response. Thus, if a cache has a response with:

Variant-Key: (fr gzip)

it could be used to satisfy the first preference. If not, responses corresponding to the other keys could be returned, or the request could be forwarded towards the origin.

#### 4.3.1. A Variant Missing From the Cache

If the selected variants-header was:

Variants: Accept-Language=(en fr de)

And a request comes in with the following headers:

Accept-Language: de;q=1.0, es;q=0.8

Then sorted-variants in Cache Behaviour is:

```
[
  ["de"]           // prefers German; will not accept English
]
```

If the cache contains responses with the following Variant-Keys:

Variant-Key: (fr)  
Variant-Key: (en)

Then the cache needs to forward the request to the origin server, since Variants indicates that "de" is available, and that is acceptable to the client.

#### 4.3.2. Variants That Don't Overlap the Client's Request

If the selected variants-header was:

Variants: Accept-Language=(en fr de)

And a request comes in with the following headers:

Accept-Language: es;q=1.0, ja;q=0.8

Then sorted-variants in Cache Behaviour are:

```
[  
  ["en"]  
]
```

This allows the cache to return a "Variant-Key: en" response even though it's not in the set the client prefers.

### 5. Origin Server Behaviour

Origin servers that wish to take advantage of Variants will need to generate both the Variants (Section 2) and Variant-Key (Section 3) header fields in all cacheable responses for a given resource. If either is omitted and the response is stored, it will have the effect of disabling caching for that resource until it is no longer stored (e.g., it expires, or is evicted).

Likewise, origin servers will need to assure that the members of both header field values are in the same order and have the same length, since discrepancies will cause caches to avoid using the responses they occur in.

The value of the Variants header should be relatively stable for a given resource over time; when it changes, it can have the effect of invalidating previously stored responses.

As per Section 2.1, the Vary header is required to be set appropriately when Variants is in use, so that caches that do not implement this specification still operate correctly.

Origin servers are advised to carefully consider which content negotiation mechanisms to enumerate in Variants; if a mechanism is



not supported by a receiving cache, it will "downgrade" to Vary handling, which can negatively impact cache efficiency.

### 5.1. Examples

The operation of Variants is illustrated by the examples below.

#### 5.1.1. Single Variant

Given a request/response pair:

```
GET /clancy HTTP/1.1
Host: www.example.com
Accept-Language: en;q=1.0, fr;q=0.5
```

```
HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Cache-Control: max-age=3600
Variants: Accept-Language=(en de)
Variant-Key: (en)
Vary: Accept-Language
Transfer-Encoding: chunked
```

Upon receipt of this response, the cache knows that two representations of this resource are available, one with a language of "en", and another whose language is "de".

Subsequent requests (while this response is fresh) will cause the cache to either reuse this response or forward the request, depending on what the selection algorithm determines.

So, if a request with "en" in Accept-Language is received and its q-value indicates that it is acceptable, the stored response is used. A request that indicates that "de" is acceptable will be forwarded to the origin, thereby populating the cache. A cache receiving a request that indicates both languages are acceptable will use the q-value to make a determination of what response to return.

A cache receiving a request that does not list either language as acceptable (or does not contain an Accept-Language at all) will return the "en" representation (possibly fetching it from the origin), since it is listed first in the Variants list.

Note that Accept-Language is listed in Vary, to assure backwards-compatibility with caches that do not support Variants.

### 5.1.2. Multiple Variants

A more complicated request/response pair:

```
GET /murray HTTP/1.1
Host: www.example.net
Accept-Language: en;q=1.0, fr;q=0.5
Accept-Encoding: gzip, br
```

```
HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Content-Encoding: br
Variants: Accept-Language=(en jp de)
Variants: Accept-Encoding=(br gzip)
Variant-Key: (en br)
Vary: Accept-Language, Accept-Encoding
Transfer-Encoding: chunked
```

Here, the cache knows that there are two axes that the response varies upon; language and encoding. Thus, there are a total of nine possible representations for the resource (including the identity encoding), and the cache needs to consider the selection algorithms for both axes.

Upon a subsequent request, if both selection algorithms return a stored representation, it can be served from cache; otherwise, the request will need to be forwarded to origin.

### 5.1.3. Partial Coverage

Now, consider the previous example, but where only one of the Vary'd axes (encoding) is listed in Variants:

```
GET /bar HTTP/1.1
Host: www.example.net
Accept-Language: en;q=1.0, fr;q=0.5
Accept-Encoding: gzip, br
```

```
HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Content-Encoding: br
Variants: Accept-Encoding=(br gzip)
Variant-Key: (br)
Vary: Accept-Language, Accept-Encoding
Transfer-Encoding: chunked
```

Here, the cache will need to calculate a secondary cache key as per [RFC7234], Section 4.1 – but considering only Accept-Language to be in its field-value – and then continue processing Variants for the set of stored responses that the algorithm described there selects.

## 6. Defining Content Negotiation Using Variants

To be usable with Variants, proactive content negotiation mechanisms need to be specified to take advantage of it. Specifically, they:

- o MUST define a request header field that advertises the clients preferences or capabilities, whose field-name SHOULD begin with "Accept-".
- o MUST define the syntax of an available-value that will occur in Variants and Variant-Key.
- o MUST define an algorithm for selecting a result. It MUST return a list of available-values that are suitable for the request, in order of preference, given the value of the request header nominated above (or null if the request header is absent) and an available-values list from the Variants header. If the result is an empty list, it implies that the cache cannot satisfy the request.

Appendix A fulfils these requirements for some existing proactive content negotiation mechanisms in HTTP.

## 7. IANA Considerations

This specification registers the following entry in the Permanent Message Header Field Names registry established by [RFC3864]:

- o Header field name: Variants
- o Applicable protocol: http
- o Status: standard

- o Author/Change Controller: IETF
- o Specification document(s): [this document]
- o Related information:

This specification registers the following entry in the Permanent Message Header Field Names registry established by [RFC3864]:

- o Header field name: Variant-Key
- o Applicable protocol: http
- o Status: standard
- o Author/Change Controller: IETF
- o Specification document(s): [this document]
- o Related information:

## 8. Security Considerations

If the number or advertised characteristics of the representations available for a resource are considered sensitive, the Variants header by its nature will leak them.

Note that the Variants header is not a commitment to make representations of a certain nature available; the runtime behaviour of the server always overrides hints like Variants.

## 9. References

### 9.1. Normative References

- [I-D.ietf-httpbis-header-structure]  
Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-13 (work in progress), August 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4647] Phillips, A. and M. Davis, "Matching of Language Tags", BCP 47, RFC 4647, DOI 10.17487/RFC4647, September 2006, <<https://www.rfc-editor.org/info/rfc4647>>.

- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/info/rfc7234>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 9.2. Informative References

- [I-D.ietf-httpbis-client-hints] Grigorik, I., "HTTP Client Hints", draft-ietf-httpbis-client-hints-07 (work in progress), March 2019.
- [RFC2295] Holtman, K. and A. Mutz, "Transparent Content Negotiation in HTTP", RFC 2295, DOI 10.17487/RFC2295, March 1998, <<https://www.rfc-editor.org/info/rfc2295>>.
- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <<https://www.rfc-editor.org/info/rfc3864>>.

## 9.3. URIs

- [1] <https://lists.w3.org/Archives/Public/ietf-http-wg/>
- [2] <https://httpwg.github.io/>

[3] <https://github.com/httpwg/http-extensions/labels/variants>

[4] <https://github.com/mnot/variants-toy>

## Appendix A. Variants for Existing Content Negotiation Mechanisms

This appendix defines the required information to use existing proactive content negotiation mechanisms (as defined in [RFC7231], Section 5.3) with the Variants header field.

### A.1. Accept

This section defines variant handling for the Accept request header (section 5.3.2 of [RFC7231]).

The syntax of an available-value for Accept is:

accept-available-value = type "/" subtype

To perform content negotiation for Accept given a request-value and available-values:

1. Let preferred-available be an empty list.
2. Let preferred-types be a list of the types in the request-value (or the empty list if request-value is null), ordered by their weight, highest to lowest, as per Section 5.3.2 of [RFC7231] (omitting any coding with a weight of 0). If a type lacks an explicit weight, an implementation MAY assign one.
3. For each preferred-type in preferred-types:
  1. If any member of available-values matches preferred-type, using the media-range matching mechanism specified in Section 5.3.2 of [RFC7231] (which is case-insensitive), append those members of available-values to preferred-available (preserving the precedence order implied by the media ranges' specificity).
4. If preferred-available is empty, append the first member of available-values to preferred-available. This makes the first available-value the default when none of the client's preferences are available.
5. Return preferred-available.

Note that this algorithm explicitly ignores extension parameters on media types (e.g., "charset").

## A.2. Accept-Encoding

This section defines variant handling for the Accept-Encoding request header (section 5.3.4 of [RFC7231]).

The syntax of an available-value for Accept-Encoding is:

accept-encoding-available-value = content-coding / "identity"

To perform content negotiation for Accept-Encoding given a request-value and available-values:

1. Let preferred-available be an empty list.
2. Let preferred-codings be a list of the codings in the request-value (or the empty list if request-value is null), ordered by their weight, highest to lowest, as per Section 5.3.1 of [RFC7231] (omitting any coding with a weight of 0). If a coding lacks an explicit weight, an implementation MAY assign one.
3. If "identity" is not a member of preferred-codings, append "identity".
4. Append "identity" to available-values.
5. For each preferred-coding in preferred-codings:
  1. If there is a case-insensitive, character-for-character match for preferred-coding in available-values, append that member of available-values to preferred-available.
6. Return preferred-available.

Note that the unencoded variant needs to have a Variant-Key header field with a value of "identity" (as defined in Section 5.3.4 of [RFC7231]).

## A.3. Accept-Language

This section defines variant handling for the Accept-Language request header (section 5.3.5 of [RFC7231]).

The syntax of an available-value for Accept-Language is:

accept-encoding-available-value = language-range

To perform content negotiation for Accept-Language given a request-value and available-values:

1. Let preferred-available be an empty list.
2. Let preferred-langs be a list of the language-ranges in the request-value (or the empty list if request-value is null), ordered by their weight, highest to lowest, as per Section 5.3.1 of [RFC7231] (omitting any language-range with a weight of 0). If a language-range lacks a weight, an implementation MAY assign one.
3. For each preferred-lang in preferred-langs:
  1. If any member of available-values matches preferred-lang, using either the Basic or Extended Filtering scheme defined in Section 3.3 of [RFC4647], append those members of available-values to preferred-available (preserving their order).
4. If preferred-available is empty, append the first member of available-values to preferred-available. This makes the first available-value the default when none of the client's preferences are available.
5. Return preferred-available.

#### A.4. Cookie

This section defines variant handling for the Cookie request header ([RFC6265]).

This syntax of an available-value for Cookie is:

cookie-available-value = cookie-name

To perform content negotiation for Cookie given a request-value and available-values:

1. Let cookies-available be an empty list.
2. For each available-value of available-values:
  1. Parse request-value as a Cookie header field [RFC6265] and let request-cookie-value be the cookie-value corresponding to a cookie with a cookie-name that matches available-value. If no match is found, continue to the next available-value.
  2. append request-cookie-value to cookies-available.
3. Return cookies-available.



A simple example is allowing a page designed for users that aren't logged in (denoted by the "logged\_in" cookie-name) to be cached:

```
Variants: Cookie=(logged_in)
Variant-Key: (0)
Vary: Cookie
```

Here, a cache that implements Variants will only use this response to satisfy requests with "Cookie: logged\_in=0". Caches that don't implement Variants will vary the response on all Cookie headers.

Or, consider this example:

```
Variants: Cookie=(user_priority)
Variant-Key: (silver), ("bronze")
Vary: Cookie
```

Here, the "user\_priority" cookie-name allows requests from "gold" users to be separated from "silver" and "bronze" ones; this response is only served to the latter two.

It is possible to target a response to a single user; for example:

```
Variants: Cookie=(user_id)
Variant-Key: (some_person)
Vary: Cookie
```

Here, only the "some\_person" "user\_id" will have this response served to them again.

Note that if more than one cookie-name serves as a cache key, they'll need to be listed in separate Variants members, like this:

```
Variants: Cookie=(user_priority), Cookie=(user_region)
Variant-Key: (gold europe)
Vary: Cookie
```

#### Acknowledgements

This protocol is conceptually similar to, but simpler than, Transparent Content Negotiation [RFC2295]. Thanks to its authors for their inspiration.

It is also a generalisation of a Fastly VCL feature designed by Rogier 'DocWilco' Mulhuijzen.

Thanks to Hooman Beheshti, Ilya Grigorik, Leif Hedstrom, and Jeffrey Yasskin for their review and input.

Author's Address

Mark Nottingham  
Fastly

Email: [mnot@mnot.net](mailto:mnot@mnot.net)  
URI: <https://www.mnot.net/>

HTTP  
Internet-Draft  
Intended status: Standards Track  
Expires: 30 May 2021

R. Polli  
Team Digitale, Italian Government  
A. Martinez  
Red Hat  
26 November 2020

RateLimit Header Fields for HTTP  
draft-polli-ratelimit-headers-05

Abstract

This document defines the RateLimit-Limit, RateLimit-Remaining, RateLimit-Reset fields for HTTP, thus allowing servers to publish current request quotas and clients to shape their request policy and avoid being throttled out.

Note to Readers

\_RFC EDITOR: please remove this section before publication\_

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

The source code and issues list for this draft can be found at <https://github.com/ioggstream/draft-polli-ratelimit-headers> (<https://github.com/ioggstream/draft-polli-ratelimit-headers>).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 30 May 2021.

## Copyright Notice

Copyright (c) 2020 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 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 . . . . .	3
1.1. Rate-limiting and quotas . . . . .	3
1.2. Current landscape of rate-limiting headers . . . . .	4
1.2.1. Interoperability issues . . . . .	4
1.3. This proposal . . . . .	5
1.4. Goals . . . . .	5
1.5. Notational Conventions . . . . .	6
2. Expressing rate-limit policies . . . . .	6
2.1. Time window . . . . .	6
2.2. Request quota . . . . .	6
2.3. Quota policy . . . . .	7
3. Header Specifications . . . . .	8
3.1. RateLimit-Limit . . . . .	8
3.2. RateLimit-Remaining . . . . .	9
3.3. RateLimit-Reset . . . . .	9
4. Providing RateLimit headers . . . . .	10
5. Intermediaries . . . . .	11
6. Caching . . . . .	11
7. Receiving RateLimit headers . . . . .	11
8. Examples . . . . .	12
8.1. Unparameterized responses . . . . .	12
8.1.1. Throttling informations in responses . . . . .	12
8.1.2. Use in conjunction with custom headers . . . . .	13
8.1.3. Use for limiting concurrency . . . . .	13
8.1.4. Use in throttled responses . . . . .	14
8.2. Parameterized responses . . . . .	15
8.2.1. Throttling window specified via parameter . . . . .	15
8.2.2. Dynamic limits with parameterized windows . . . . .	15
8.2.3. Dynamic limits for pushing back and slowing down . . . . .	16
8.3. Dynamic limits for pushing back with Retry-After and slow down . . . . .	17
8.3.1. Missing Remaining informations . . . . .	17

8.3.2. Use with multiple windows . . . . .	18
9. Security Considerations . . . . .	19
9.1. Throttling does not prevent clients from issuing requests . . . . .	19
9.2. Information disclosure . . . . .	19
9.3. Remaining quota-units are not granted requests . . . . .	19
9.4. Reliability of RateLimit-Reset . . . . .	20
9.5. Resource exhaustion . . . . .	20
9.6. Denial of Service . . . . .	20
10. IANA Considerations . . . . .	20
10.1. RateLimit-Limit Field Registration . . . . .	21
10.2. RateLimit-Remaining Field Registration . . . . .	21
10.3. RateLimit-Reset Field Registration . . . . .	21
11. References . . . . .	21
11.1. Normative References . . . . .	21
11.2. Informative References . . . . .	22
Appendix A. Change Log . . . . .	23
Appendix B. Acknowledgements . . . . .	23
Appendix C. RateLimit headers currently used on the web . . . . .	23
Appendix D. FAQ . . . . .	24
Authors' Addresses . . . . .	27

## 1. Introduction

The widespreading of HTTP as a distributed computation protocol requires an explicit way of communicating service status and usage quotas.

This was partially addressed with the "Retry-After" header field defined in [SEMANTICS] to be returned in "429 Too Many Requests" or "503 Service Unavailable" responses.

Still, there is not a standard way to communicate service quotas so that the client can throttle its requests and prevent 4xx or 5xx responses.

### 1.1. Rate-limiting and quotas

Servers use quota mechanisms to avoid systems overload, to ensure an equitable distribution of computational resources or to enforce other policies - eg. monetization.

A basic quota mechanism limits the number of acceptable requests in a given time window, eg. 10 requests per second.

When quota is exceeded, servers usually do not serve the request replying instead with a "4xx" HTTP status code (eg. 429 or 403) or adopt more aggressive policies like dropping connections.

Quotas may be enforced on different basis (eg. per user, per IP, per geographic area, ..) and at different levels. For example, an user may be allowed to issue:

- \* 10 requests per second;
- \* limited to 60 request per minute;
- \* limited to 1000 request per hour.

Moreover system metrics, statistics and heuristics can be used to implement more complex policies, where the number of acceptable request and the time window are computed dynamically.

## 1.2. Current landscape of rate-limiting headers

To help clients throttling their requests, servers may expose the counters used to evaluate quota policies via HTTP header fields.

Those response headers may be added by HTTP intermediaries such as API gateways and reverse proxies.

On the web we can find many different rate-limit headers, usually containing the number of allowed requests in a given time window, and when the window is reset.

The common choice is to return three headers containing:

- \* the maximum number of allowed requests in the time window;
- \* the number of remaining requests in the current window;
- \* the time remaining in the current window expressed in seconds or as a timestamp;

### 1.2.1. Interoperability issues

A major interoperability issue in throttling is the lack of standard headers, because:

- \* each implementation associates different semantics to the same header field names;
- \* header field names proliferates.

Client applications interfacing with different servers may thus need to process different headers, or the very same application interface that sits behind different reverse proxies may reply with different throttling headers.

### 1.3. This proposal

This proposal defines syntax and semantics for the following fields:

- \* "RateLimit-Limit": containing the requests quota in the time window;
- \* "RateLimit-Remaining": containing the remaining requests quota in the current window;
- \* "RateLimit-Reset": containing the time remaining in the current window, specified in seconds.

The behavior of "RateLimit-Reset" is compatible with the "delta-seconds" notation of "Retry-After".

The fields definition allows to describe complex policies, including the ones using multiple and variable time windows and dynamic quotas, or implementing concurrency limits.

### 1.4. Goals

The goals of this proposal are:

1. Standardizing the names and semantic of rate-limit headers;
2. Improve resiliency of HTTP infrastructures simplifying the enforcement and the adoption of rate-limit headers;
3. Simplify API documentation avoiding expliciting rate-limit fields semantic in documentation.

The goals do not include:

Authorization: The rate-limit headers described here are not meant to support authorization or other kinds of access controls.

Throttling scope: This specification does not cover the throttling scope, that may be the given resource-target, its parent path or the whole Origin [RFC6454] section 7.

Response status code: The rate-limit headers may be returned in both

Successful and non Successful responses. This specification does not cover whether non Successful responses count on quota usage.

Throttling policy: This specification does not mandate a specific throttling policy. The values published in the headers, including the window size, can be statically or dynamically evaluated.

Service Level Agreement: Conveyed quota hints do not imply any service guarantee. Server is free to throttle respectful clients under certain circumstances.

## 1.5. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the Augmented BNF defined in [RFC5234] and updated by [RFC7405] along with the "#rule" extension defined in Section 7 of [MESSAGING].

The term Origin is to be interpreted as described in [RFC6454] section 7.

The "delta-seconds" rule is defined in [CACHING] section 1.2.1.

## 2. Expressing rate-limit policies

### 2.1. Time window

Rate limit policies limit the number of acceptable requests in a given time window.

A time window is expressed in seconds, using the following syntax:

time-window = delta-seconds

Subsecond precision is not supported.

### 2.2. Request quota

The request-quota is a value associated to the maximum number of requests that the server is willing to accept from one or more clients on a given basis (originating IP, authenticated user, geographical, ..) during a "time-window" as defined in Section 2.1.



The "request-quota" is expressed in "quota-units" and has the following syntax:

```
request-quota = quota-units
quota-units = 1*DIGIT
```

The "request-quota" SHOULD match the maximum number of acceptable requests.

The "request-quota" MAY differ from the total number of acceptable requests when weight mechanisms, bursts, or other server policies are implemented.

If the "request-quota" does not match the maximum number of acceptable requests the relation with that SHOULD be communicated out-of-band.

Example: A server could

- \* count once requests like "/books/{id}"

- \* count twice search requests like "/books?author=Camilleri"

so that we have the following counters

```
GET /books/123                ; request-quota=4, remaining: 3, status=200
GET /books?author=Camilleri    ; request-quota=4, remaining: 1, status=200
GET /books?author=Eco          ; request-quota=4, remaining: 0, status=429
```

### 2.3. Quota policy

This specification allows describing a quota policy with the following syntax:

```
quota-policy = request-quota; "w" "=" time-window
              *( OWS ";" OWS quota-comment)
quota-comment = token "=" (token / quoted-string)
```

quota-policy parameters like "w" and quota-comment tokens MUST NOT occur multiple times within the same quota-policy.

An example policy of 100 quota-units per minute.

```
100;w=60
```

Two examples of providing further details via custom parameters in "quota-comments".

```
100;w=60;comment="fixed window"
12;w=1;burst=1000;policy="leaky bucket"
```

### 3. Header Specifications

The following "RateLimit" response fields are defined

#### 3.1. RateLimit-Limit

The "RateLimit-Limit" response field indicates the "request-quota" associated to the client in the current "time-window".

If the client exceeds that limit, it MAY not be served.

The header value is

```
RateLimit-Limit = expiring-limit [ , 1#quota-policy ]
expiring-limit = request-quota
```

The "expiring-limit" value MUST be set to the "request-quota" that is closer to reach its limit.

The "quota-policy" is defined in Section 2.3, and its values are informative.

```
RateLimit-Limit: 100
```

A "time-window" associated to "expiring-limit" can be communicated via an optional "quota-policy" value, like shown in the following example

```
RateLimit-Limit: 100, 100;w=10
```

If the "expiring-limit" is not associated to a "time-window", the "time-window" MUST either be:

- \* inferred by the value of "RateLimit-Reset" at the moment of the reset, or
- \* communicated out-of-band (eg. in the documentation).

Policies using multiple quota limits MAY be returned using multiple "quota-policy" items, like shown in the following two examples:

```
RateLimit-Limit: 10, 10;w=1, 50;w=60, 1000;w=3600, 5000;w=86400
RateLimit-Limit: 10, 10;w=1;burst=1000, 1000;w=3600
```

This header MUST NOT occur multiple times and can be sent in a trailer section.

### 3.2. RateLimit-Remaining

The "RateLimit-Remaining" response field indicates the remaining "quota-units" defined in Section 2.2 associated to the client.

The header value is

RateLimit-Remaining = quota-units

This header MUST NOT occur multiple times and can be sent in a trailer section.

Clients MUST NOT assume that a positive "RateLimit-Remaining" value is a guarantee of being served.

A low "RateLimit-Remaining" value is like a yellow traffic-light: the red light may arrive suddenly.

One example of "RateLimit-Remaining" use is below.

RateLimit-Remaining: 50

### 3.3. RateLimit-Reset

The "RateLimit-Reset" response field indicates either

- \* the number of seconds until the quota resets.

The header value is

RateLimit-Reset = delta-seconds

The delta-seconds format is used because:

- \* it does not rely on clock synchronization and is resilient to clock adjustment and clock skew between client and server (see [SEMANTICS] Section 4.1.1.1);
- \* it mitigates the risk related to thundering herd when too many clients are serviced with the same timestamp.

This header MUST NOT occur multiple times and can be sent in a trailer section.

An example of "RateLimit-Reset" use is below.

RateLimit-Reset: 50

The client MUST NOT assume that all its "request-quota" will be restored after the moment referenced by "RateLimit-Reset". The server MAY arbitrarily alter the "RateLimit-Reset" value between subsequent requests eg. in case of resource saturation or to implement sliding window policies.

#### 4. Providing RateLimit headers

A server MAY use one or more "RateLimit" response fields defined in this document to communicate its quota policies.

The returned values refers to the metrics used to evaluate if the current request respects the quota policy and MAY not apply to subsequent requests.

Example: a successful response with the following fields

```
RateLimit-Limit: 10
RateLimit-Remaining: 1
RateLimit-Reset: 7
```

does not guarantee that the next request will be successful. Server metrics may be subject to other conditions like the one shown in the example from Section 2.2.

A server MAY return "RateLimit" response fields independently of the response status code. This includes throttled responses.

If a response contains both the "Retry-After" and the "RateLimit-Reset" fields, the value of "RateLimit-Reset" SHOULD reference the same point in time as "Retry-After".

When using a policy involving more than one "time-window", the server MUST reply with the "RateLimit" headers related to the window with the lower "RateLimit-Remaining" values.

Under certain conditions, a server MAY artificially lower "RateLimit" field values between subsequent requests, eg. to respond to Denial of Service attacks or in case of resource saturation.

Servers usually establish whether the request is in-quota before creating a response, so the RateLimit field values should be already available in that moment. Nonetheless servers MAY decide to send the "RateLimit" fields in a trailer section.

## 5. Intermediaries

This section documents the considerations advised in Section 15.3.3 of [SEMANTICS].

An intermediary that is not part of the originating service infrastructure and is not aware of the quota-policy semantic used by the Origin Server SHOULD NOT alter the RateLimit fields' values in such a way as to communicate a more permissive quota-policy; this includes removing the RateLimit fields.

An intermediary MAY alter the RateLimit fields in such a way as to communicate a more restrictive quota-policy when:

- \* it is aware of the quota-unit semantic used by the Origin Server;
- \* it implements this specification and enforces a quota-policy which is more restrictive than the one conveyed in the fields.

An intermediary SHOULD forward a request even when presuming that it might not be serviced; the service returning the RateLimit fields is the sole responsible of enforcing the communicated quota-policy, and it is always free to service incoming requests.

This specification does not mandate any behavior on intermediaries respect to retries, nor requires that intermediaries have any role in respecting quota-policies. For example, it is legitimate for a proxy to retransmit a request without notifying the client, and thus consuming quota-units.

## 6. Caching

As is the ordinary case for HTTP caching ([RFC7234]), a response with RateLimit fields might be cached and re-used for subsequent requests. A cached RateLimit response, does not modify quota counters but could contain stale information. Clients interested in determining the freshness of the RateLimit fields could rely on fields such as "Date" and on the "window" value of a "quota-policy".

## 7. Receiving RateLimit headers

A client MUST process the received "RateLimit" headers.

A client MUST validate the values received in the "RateLimit" headers before using them and check if there are significant discrepancies with the expected ones. This includes a "RateLimit-Reset" moment too far in the future or a "request-quota" too high.

Malformed "RateLimit" headers MAY be ignored.

A client SHOULD NOT exceed the "quota-units" expressed in "RateLimit-Remaining" before the "time-window" expressed in "RateLimit-Reset".

A client MAY still probe the server if the "RateLimit-Reset" is considered too high.

The value of "RateLimit-Reset" is generated at response time: a client aware of a significant network latency MAY behave accordingly and use other informations (eg. the "Date" response header, or otherwise gathered metrics) to better estimate the "RateLimit-Reset" moment intended by the server.

The "quota-policy" values and comments provided in "RateLimit-Limit" are informative and MAY be ignored.

If a response contains both the "RateLimit-Reset" and "Retry-After" fields, the "Retry-After" header field MUST take precedence and the "RateLimit-Reset" field MAY be ignored.

## 8. Examples

### 8.1. Unparameterized responses

#### 8.1.1. Throttling informations in responses

The client exhausted its request-quota for the next 50 seconds. The "time-window" is communicated out-of-band or inferred by the header values.

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 100
Ratelimit-Remaining: 0
Ratelimit-Reset: 50

{"hello": "world"}
```

### 8.1.2. Use in conjunction with custom headers

The server uses two custom headers, namely "acme-RateLimit-DayLimit" and "acme-RateLimit-HourLimit" to expose the following policy:

- \* 5000 daily quota-units;
- \* 1000 hourly quota-units.

The client consumed 4900 quota-units in the first 14 hours.

Despite the next hourly limit of 1000 quota-units, the closest limit to reach is the daily one.

The server then exposes the "RateLimit-\*" headers to inform the client that:

- \* it has only 100 quota-units left;
- \* the window will reset in 10 hours.

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
acme-RateLimit-DayLimit: 5000
acme-RateLimit-HourLimit: 1000
RateLimit-Limit: 5000
RateLimit-Remaining: 100
RateLimit-Reset: 36000

{"hello": "world"}
```

### 8.1.3. Use for limiting concurrency

Throttling headers may be used to limit concurrency, advertising limits that are lower than the usual ones in case of saturation, thus increasing availability.

The server adopted a basic policy of 100 quota-units per minute, and in case of resource exhaustion adapts the returned values reducing both "RateLimit-Limit" and "RateLimit-Remaining".

After 2 seconds the client consumed 40 quota-units

Request:

GET /items/123

Response:

HTTP/1.1 200 Ok  
Content-Type: application/json  
RateLimit-Limit: 100  
RateLimit-Remaining: 60  
RateLimit-Reset: 58

{"elapsed": 2, "issued": 40}

At the subsequent request - due to resource exhaustion - the server advertises only "RateLimit-Remaining: 20".

Request:

GET /items/123

Response:

HTTP/1.1 200 Ok  
Content-Type: application/json  
RateLimit-Limit: 100  
RateLimit-Remaining: 20  
RateLimit-Reset: 56

{"elapsed": 4, "issued": 41}

#### 8.1.4. Use in throttled responses

A client exhausted its quota and the server throttles the request sending the "Retry-After" response header field.

In this example, the values of "Retry-After" and "RateLimit-Reset" reference the same moment, but this is not a requirement.

The "429 Too Many Requests" HTTP status code is just used as an example.

Request:

GET /items/123

Response:



```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
Date: Mon, 05 Aug 2019 09:27:00 GMT
Retry-After: Mon, 05 Aug 2019 09:27:05 GMT
RateLimit-Reset: 5
RateLimit-Limit: 100
Ratelimit-Remaining: 0
```

```
{
  "title": "Too Many Requests",
  "status": 429,
  "detail": "You have exceeded your quota"
}
```

## 8.2. Parameterized responses

### 8.2.1. Throttling window specified via parameter

The client has 99 "quota-units" left for the next 50 seconds. The "time-window" is communicated by the "w" parameter, so we know the throughput is 100 "quota-units" per minute.

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 100, 100;w=60
Ratelimit-Remaining: 99
Ratelimit-Reset: 50
```

```
{"hello": "world"}
```

### 8.2.2. Dynamic limits with parameterized windows

The policy conveyed by "RateLimit-Limit" states that the server accepts 100 quota-units per minute.

To avoid resource exhaustion, the server artificially lowers the actual limits returned in the throttling headers.

The "RateLimit-Remaining" then advertises only 9 quota-units for the next 50 seconds to slow down the client.

Note that the server could have lowered even the other values in "RateLimit-Limit": this specification does not mandate any relation between the field values contained in subsequent responses.

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 10, 100;w=60
Ratelimit-Remaining: 9
Ratelimit-Reset: 50

{
  "status": 200,
  "detail": "Just slow down without waiting."
}
```

### 8.2.3. Dynamic limits for pushing back and slowing down

Continuing the previous example, let's say the client waits 10 seconds and performs a new request which, due to resource exhaustion, the server rejects and pushes back, advertising "RateLimit-Remaining: 0" for the next 20 seconds.

The server advertises a smaller window with a lower limit to slow down the client for the rest of its original window after the 20 seconds elapse.

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
RateLimit-Limit: 0, 15;w=20
Ratelimit-Remaining: 0
Ratelimit-Reset: 20

{
  "status": 429,
  "detail": "Wait 20 seconds, then slow down!"
}
```

### 8.3. Dynamic limits for pushing back with Retry-After and slow down

Alternatively, given the same context where the previous example starts, we can convey the same information to the client via the Retry-After header, with the advantage that the server can now specify the policy's nominal limit and window that will apply after the reset, ie. assuming the resource exhaustion is likely to be gone by then, so the advertised policy does not need to be adjusted, yet we managed to stop requests for a while and slow down the rest of the current window.

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
Retry-After: 20
RateLimit-Limit: 15, 100;w=60
Ratelimit-Remaining: 15
Ratelimit-Reset: 40
```

```
{
  "status": 429,
  "detail": "Wait 20 seconds, then slow down!"
}
```

Note that in this last response the client is expected to honor the "Retry-After" header and perform no requests for the specified amount of time, whereas the previous example would not force the client to stop requests before the reset time is elapsed, as it would still be free to query again the server even if it is likely to have the request rejected.

#### 8.3.1. Missing Remaining informations

The server does not expose "RateLimit-Remaining" values, but resets the limit counter every second.

It communicates to the client the limit of 10 quota-units per second always returning the couple "RateLimit-Limit" and "RateLimit-Reset".

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 10
Ratelimit-Reset: 1
```

```
{"first": "request"}
```

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 10
Ratelimit-Reset: 1
```

```
{"second": "request"}
```

#### 8.3.2. Use with multiple windows

This is a standardized way of describing the policy detailed in Section 8.1.2:

- \* 5000 daily quota-units;
- \* 1000 hourly quota-units.

The client consumed 4900 quota-units in the first 14 hours.

Despite the next hourly limit of 1000 quota-units, the closest limit to reach is the daily one.

The server then exposes the "RateLimit" headers to inform the client that:

- \* it has only 100 quota-units left;
- \* the window will reset in 10 hours;
- \* the "expiring-limit" is 5000.

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
RateLimit-Limit: 5000, 1000;w=3600, 5000;w=86400
RateLimit-Remaining: 100
RateLimit-Reset: 36000
```

```
{"hello": "world"}
```

## 9. Security Considerations

### 9.1. Throttling does not prevent clients from issuing requests

This specification does not prevent clients to make over-quota requests.

Servers should always implement mechanisms to prevent resource exhaustion.

### 9.2. Information disclosure

Servers should not disclose operational capacity informations that can be used to saturate its resources.

While this specification does not mandate whether non 2xx responses consume quota, if 401 and 403 responses count on quota a malicious client could probe the endpoint to get traffic informations of another user.

As intermediaries might retransmit requests and consume quota-units without prior knowledge of the User Agent, RateLimit headers might reveal the existence of an intermediary to the User Agent.

### 9.3. Remaining quota-units are not granted requests

"RateLimit-\*" headers convey hints from the server to the clients in order to avoid being throttled out.

Clients MUST NOT consider the "quota-units" returned in "RateLimit-Remaining" as a service level agreement.

In case of resource saturation, the server MAY artificially lower the returned values or not serve the request anyway.

#### 9.4. Reliability of RateLimit-Reset

Consider that "request-quota" may not be restored after the moment referenced by "RateLimit-Reset", and the "RateLimit-Reset" value should not be considered fixed nor constant.

Subsequent requests may return an higher "RateLimit-Reset" value to limit concurrency or implement dynamic or adaptive throttling policies.

#### 9.5. Resource exhaustion

When returning "RateLimit-Reset" you must be aware that many throttled clients may come back at the very moment specified.

This is true for "Retry-After" too.

For example, if the quota resets every day at "18:00:00" and your server returns the "RateLimit-Reset" accordingly

```
Date: Tue, 15 Nov 1994 08:00:00 GMT
RateLimit-Reset: 36000
```

there's a high probability that all clients will show up at "18:00:00".

This could be mitigated adding some jitter to the field-value.

#### 9.6. Denial of Service

"RateLimit" fields may assume unexpected values by chance or purpose. For example, an excessively high "RateLimit-Remaining" value may be:

- \* used by a malicious intermediary to trigger a Denial of Service attack or consume client resources boosting its requests;
- \* passed by a misconfigured server;

or an high "RateLimit-Reset" value could inhibit clients to contact the server.

Clients MUST validate the received values to mitigate those risks.

#### 10. IANA Considerations

### 10.1. RateLimit-Limit Field Registration

This section registers the "RateLimit-Limit" field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]).

Field name: "RateLimit-Limit"

Status: permanent

Specification document(s): Section 3.1 of this document

### 10.2. RateLimit-Remaining Field Registration

This section registers the "RateLimit-Remaining" field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]).

Field name: "RateLimit-Remaining"

Status: permanent

Specification document(s): Section 3.2 of this document

### 10.3. RateLimit-Reset Field Registration

This section registers the "RateLimit-Reset" field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]).

Field name: "RateLimit-Reset"

Status: permanent

Specification document(s): Section 3.3 of this document

## 11. References

### 11.1. Normative References

[CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/info/rfc7234>>.

[MESSAGING] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <<https://www.rfc-editor.org/info/rfc7405>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [SEMANTICS] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [UNIX] The Open Group, ., "The Single UNIX Specification, Version 2 - 6 Vol Set for UNIX 98", February 1997.

## 11.2. Informative References

- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <<https://www.rfc-editor.org/info/rfc6585>>.
- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/info/rfc7234>>.



## Appendix A. Change Log

RFC EDITOR PLEASE DELETE THIS SECTION.

## Appendix B. Acknowledgements

Thanks to Willi Schoenborn, Alejandro Martinez Ruiz, Alessandro Ranellucci, Amos Jeffries, Martin Thomson, Erik Wilde and Mark Nottingham for being the initial contributors of these specifications. Kudos to the first community implementors: Aapo Talvensaari, Nathan Friedly and Sanyam Dogra.

## Appendix C. RateLimit headers currently used on the web

RFC EDITOR PLEASE DELETE THIS SECTION.

Commonly used header field names are:

- \* "X-RateLimit-Limit", "X-RateLimit-Remaining", "X-RateLimit-Reset";
- \* "X-Rate-Limit-Limit", "X-Rate-Limit-Remaining", "X-Rate-Limit-Reset".

There are variants too, where the window is specified in the header field name, eg:

- \* "x-ratelimit-limit-minute", "x-ratelimit-limit-hour", "x-ratelimit-limit-day"
- \* "x-ratelimit-remaining-minute", "x-ratelimit-remaining-hour", "x-ratelimit-remaining-day"

Here are some interoperability issues:

- \* "X-RateLimit-Remaining" references different values, depending on the implementation:
  - seconds remaining to the window expiration
  - milliseconds remaining to the window expiration
  - seconds since UTC, in UNIX Timestamp
  - a datetime, either "IMF-fixdate" [SEMANTICS] or [RFC3339]
- \* different headers, with the same semantic, are used by different implementers:

- X-RateLimit-Limit and X-Rate-Limit-Limit
- X-RateLimit-Remaining and X-Rate-Limit-Remaining
- X-RateLimit-Reset and X-Rate-Limit-Reset

The semantic of RateLimit-Remaining depends on the windowing algorithm. A sliding window policy for example may result in having a ratelimit-remaining value related to the ratio between the current and the maximum throughput. Eg.

```
RateLimit-Limit: 12, 12;w=1
RateLimit-Remaining: 6           ; using 50% of throughput, that is 6 units/s
RateLimit-Reset: 1
```

If this is the case, the optimal solution is to achieve

```
RateLimit-Limit: 12, 12;w=1
RateLimit-Remaining: 1           ; using 100% of throughput, that is 12 units/s
RateLimit-Reset: 1
```

At this point you should stop increasing your request rate.

#### Appendix D. FAQ

1. Why defining standard headers for throttling?

To simplify enforcement of throttling policies.

2. Can I use RateLimit-\* in throttled responses (eg with status code 429)?

Yes, you can.

3. Are those specs tied to RFC 6585?

No. [RFC6585] defines the "429" status code and we use it just as an example of a throttled request, that could instead use even 403 or whatever status code.

4. Why don't pass the throttling scope as a parameter?

After a discussion on a similar thread (<https://github.com/httpwg/http-core/pull/317#issuecomment-585868767>) we will probably add a new "RateLimit-Scope" header to this spec.

I'm open to suggestions: comment on this issue  
(<https://github.com/ioggstream/draft-polli-ratelimit-headers/issues/70>)

5. Why using delta-seconds instead of a UNIX Timestamp? Why not using subsecond precision?

Using delta-seconds aligns with "Retry-After", which is returned in similar contexts, eg on 429 responses.

delta-seconds as defined in [CACHING] section 1.2.1 clarifies some parsing rules too.

Timestamps require a clock synchronization protocol (see [SEMANTICS] section 4.1.1.1). This may be problematic (eg. clock adjustment, clock skew, failure of hardcoded clock synchronization servers, IoT devices, ..). Moreover timestamps may not be monotonically increasing due to clock adjustment. See Another NTP client failure story (<https://community.ntppool.org/t/another-ntp-client-failure-story/1014/>)

We did not use subsecond precision because:

- \* that is more subject to system clock correction like the one implemented via the adjtimex() Linux system call;
- \* response-time latency may not make it worth. A brief discussion on the subject is on the httpwg ml (<https://lists.w3.org/Archives/Public/ietf-http-wg/2019JulSep/0202.html>)
- \* almost all rate-limit headers implementations do not use it.

6. Why not support multiple quota remaining?

While this might be of some value, my experience suggests that overly-complex quota implementations results in lower effectiveness of this policy. This spec allows the client to easily focusing on RateLimit-Remaining and RateLimit-Reset.

7. Shouldn't I limit concurrency instead of request rate?

You can use this specification to limit concurrency at the HTTP level (see {#use-for-limiting-concurrency}) and help clients to shape their requests avoiding being throttled out.

A problematic way to limit concurrency is connection dropping, especially when connections are multiplexed (eg. HTTP/2) because this results in unserved client requests, which is something we want to avoid.

A semantic way to limit concurrency is to return 503 + Retry-After in case of resource saturation (eg. thrashing, connection queues too long, Service Level Objectives not meet, ..). Saturation conditions can be either dynamic or static: all this is out of the scope for the current document.

8. Do a positive value of "RateLimit-Remaining" imply any service guarantee for my future requests to be served?

No. The returned values were used to decide whether to serve or not the current request and do not imply any guarantee that future requests will be successful.

Instead they help to understand when future requests will probably be throttled. A low value for "RateLimit-Remaining" should be interpreted as a yellow traffic-light for either the number of requests issued in the "time-window" or the request throughput.

9. Is the quota-policy definition Section 2.3 too complex?

You can always return the simplest form of the 3 headers

```
RateLimit-Limit: 100
RateLimit-Remaining: 50
RateLimit-Reset: 60
```

The key runtime value is the first element of the list: "expiring-limit", the others "quota-policy" are informative. So for the following header:

```
RateLimit-Limit: 100, 100;w=60;burst=1000;comment="sliding window", 5000;w=3600;burst=0;comment="fixed window"
```

the key value is the one referencing the lowest limit: "100"

1. Can we use shorter names? Why don't put everything in one header?

The most common syntax we found on the web is "X-RateLimit-\*" and when starting this I-D we opted for it (<https://github.com/ioggstream/draft-polli-ratelimit-headers/issues/34#issuecomment-519366481>)

The basic form of those headers is easily parseable, even by implementors processing responses using technologies like dynamic interpreter with limited syntax.

Using a single header complicates parsing and takes a significantly different approach from the existing ones: this can limit adoption.

1. Why don't mention connections?

Beware of the term "connection": - it is just one possible saturation cause. Once you go that path you will expose other infrastructural details (bandwidth, CPU, .. see Section 9.2) and complicate client compliance; - it is an infrastructural detail defined in terms of server and network rather than the consumed service. This specification protects the services first, and then the infrastructures through client cooperation (see Section 9.1). RateLimit headers enable sending on the same connection different limit values on each response, depending on the policy scope (eg. per-user, per-custom-key, ..)

2. Can intermediaries alter RateLimit fields?

Generally, they should not because it might result in unserved requests. There are reasonable use cases for intermediaries mangling RateLimit fields though, e.g. when they enforce stricter quota-policies, or when they are an active component of the service. In those case we will consider them as part of the originating infrastructure.

Authors' Addresses

Roberto Polli  
Team Digitale, Italian Government

Email: robipolli@gmail.com

Alejandro Martinez Ruiz  
Red Hat

Email: amr@redhat.com