HTTP Working Group
Internet-Draft
Obsoletes: 7234 (if approved)
Intended status: Standards Track
Expires: January 13, 2021

R. Fielding, Ed.
Adobe
M. Nottingham, Ed.
Fastly
J. F. Reschke, Ed.
greenbytes

July 12, 2020

HTTP Caching
draft-ietf-httpbis-cache-10

Abstract

The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems. This document defines HTTP caches and the associated header fields that control cache behavior or indicate cacheable response messages.

This document obsoletes RFC 7234.

Editorial Note

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

Discussion of this draft takes place on the HTTP working group mailing list (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 code and issues list for this draft can be found at <https://github.com/httpwg/http-core>.

The changes in this draft are summarized in Appendix C.11.

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 13, 2021.

Copyright Notice

Table of Contents

1. Introduction

The Hypertext Transfer Protocol (HTTP) is a stateless application-level request/response protocol that uses extensible semantics and self-descriptive messages for flexible interaction with network-based hypertext information systems. HTTP is defined by a series of documents that collectively form the HTTP/1.1 specification:

- "HTTP Semantics" [Semantics]

- "HTTP Caching" (this document)

- "HTTP/1.1 Messaging" [Messaging]

HTTP is typically used for distributed information systems, where performance can be improved by the use of response caches. This document defines aspects of HTTP related to caching and reusing response messages.

An HTTP cache is a local store of response messages and the subsystem that controls storage, retrieval, and deletion of messages in it. A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server MAY employ a cache, though a cache cannot be used by a server that is acting as a tunnel.

A shared cache is a cache that stores responses to be reused by more than one user; shared caches are usually (but not always) deployed as a part of an intermediary. A private cache, in contrast, is dedicated to a single user; often, they are deployed as a component of a user agent.

The goal of caching in HTTP is to significantly improve performance by reusing a prior response message to satisfy a current request. A stored response is considered "fresh", as defined in Section 4.2, if the response can be reused without "validation" (checking with the origin server to see if the cached response remains valid for this request). A fresh response can therefore reduce both latency and network overhead each time it is reused. When a cached response is not fresh, it might still be reusable if it can be freshened by validation (Section 4.3) or if the origin is unavailable (Section 4.2.4).

This document obsoletes RFC 7234, with the changes being summarized in Appendix B.

1.1. Requirements Notation

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.

Conformance criteria and considerations regarding error handling are defined in Section 3 of [Semantics].

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], extended with the notation for case-sensitivity in strings defined in [RFC7405].

It also uses a list extension, defined in Section 5.5 of [Semantics], that allows for compact definition of comma-separated lists using a '#' operator (similar to how the '*' operator indicates repetition). Appendix A shows the collected grammar with all list operators expanded to standard ABNF notation.

The following core rules are included by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), HTAB (horizontal tab), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), and VCHAR (any visible [USASCII] character).

The rules below are defined in [Semantics]:

```
HTTP-date     = <HTTP-date, see [Semantics], Section 5.4.1.5>
OWS           = <OWS, see [Semantics], Section 1.2.1>
field-name    = <field-name, see [Semantics], Section 5.3>
quoted-string = <quoted-string, see [Semantics], Section 5.4.1.2>
token         = <token, see [Semantics], Section 5.4.1.1>
```

1.3. Delta Seconds

The delta-seconds rule specifies a non-negative integer, representing time in seconds.

```
delta-seconds  = 1*DIGIT
```

A recipient parsing a delta-seconds value and converting it to binary form ought to use an arithmetic type of at least 31 bits of non-negative integer range. If a cache receives a delta-seconds value greater than the greatest integer it can represent, or if any of its subsequent calculations overflows, the cache MUST consider the value to be either 2147483648 (2^31) or the greatest positive integer it can conveniently represent.

> *Note:* The value 2147483648 is here for historical reasons, effectively represents infinity (over 68 years), and does not need to be stored in binary form; an implementation could produce it as a canned string if any overflow occurs, even if the calculations are performed with an arithmetic type incapable of directly representing that number. What matters here is that an overflow be detected and not treated as a negative value in later calculations.

2. Overview of Cache Operation

Proper cache operation preserves the semantics of HTTP transfers ([Semantics]) while reducing the transfer of information already held in the cache. Although caching is an entirely OPTIONAL feature of HTTP, it can be assumed that reusing a cached response is desirable and that such reuse is the default behavior when no requirement or local configuration prevents it. Therefore, HTTP cache requirements are focused on preventing a cache from either storing a non-reusable

response or reusing a stored response inappropriately, rather than mandating that caches always store and reuse particular responses.

The base cache key consists of the request method and target URI used to retrieve the stored response; the method determines under which circumstances that response can be used to satisfy a request. However, many HTTP caches in common use today only cache GET responses, and therefore only use the URI as the cache key,

forwarding other methods.

If a request target is subject to content negotiation, the cache might store multiple responses for it. Caches differentiate these responses by incorporating values of the original request's selecting header fields into the cache key as well, as per Section 4.1.

Furthermore, caches might incorporate additional material into the cache key. For example, user agent caches might include the referring site's identity, thereby "double keying" the cache to avoid some privacy risks (see Section 7.2).

Most commonly, caches store the successful result of a retrieval request: i.e., a 200 (OK) response to a GET request, which contains a representation of the target resource (Section 8.3.1 of [Semantics]). However, it is also possible to store redirects, negative results (e.g., 404 (Not Found)), incomplete results (e.g., 206 (Partial Content)), and responses to methods other than GET if the method's definition allows such caching and defines something suitable for use as a cache key.

A cache is disconnected when it cannot contact the origin server or otherwise find a forward path for a given request. A disconnected cache can serve stale responses in some circumstances (Section 4.2.4).

3. Storing Responses in Caches

A cache MUST NOT store a response to a request unless:

- the request method is understood by the cache;
- the response status code is final (see Section 10 of [Semantics]);
- if the response status code is 206 or 304, or the "must-understand" cache directive (see Section 5.2) is present: the cache understands the response status code;
- the "no-store" cache directive is not present in the response (see Section 5.2);

- if the cache is shared: the "private" response directive is either

not present or allows a modified response to be stored by a shared cache; see Section 5.2.2.7);

- if the cache is shared: the Authorization header field is not present in the request (see Section 9.5.3 of [Semantics]) or a response directive is present that explicitly allows shared caching (see Section 3.3); and,

- the response contains at least one of:

  * a public response directive (see Section 5.2.2.6);

  * a private response directive, if the cache is not shared (see Section 5.2.2.7);

  * an Expires header field (see Section 5.3);

  * a max-age response directive (see Section 5.2.2.9);

  * if the cache is shared, an s-maxage response directive (see Section 5.2.2.10);

  * a Cache Control Extension that allows it to be cached (see Section 5.2.3); or,

  * a status code that is defined as heuristically cacheable (see Section 4.2.2).

Note that any of the requirements listed above can be overridden by a cache-control extension; see Section 5.2.3.

In this context, a cache has "understood" a request method or a response status code if it recognizes it and implements all specified caching-related behavior.

Note that, in normal operation, some caches will not store a response that has neither a cache validator nor an explicit expiration time, as such responses are not usually useful to store. However, caches are not prohibited from storing such responses.

3.1. Storing Header and Trailer Fields

Caches MUST include all received header fields - including unrecognised ones - when storing a response; this assures that new HTTP header fields can be successfully deployed. However, the following exceptions are made:

- The Connection header field and fields whose names are listed in it are required by Section 9.1 of [Messaging] to be removed before forwarding the message. This MAY be implemented by doing so before storage.

- Likewise, some fields' semantics require them to be removed before forwarding the message, and this MAY be implemented by doing so before storage; see Section 9.1 of [Messaging] for some examples.

- Header fields that are specific to a client's proxy configuration MUST NOT be stored, unless the cache incorporates the identity of the proxy into the cache key. Effectively, this is limited to Proxy-Authenticate (Section 11.3.2 of [Semantics]), Proxy-Authentication-Info (Section 11.3.4 of [Semantics]), and Proxy-Authorization (Section 9.5.4 of [Semantics]).

Caches MAY either store trailer fields separately from header fields, or discard them. Caches MUST NOT combine trailer fields with header fields.

3.2. Storing Incomplete Responses

If the request method is GET, the response status code is 200 (OK), and the entire response header section has been received, a cache MAY store a response body that is not complete (Section 2.1 of [Semantics]) if the stored response is recorded as being incomplete. Likewise, a 206 (Partial Content) response MAY be stored as if it were an incomplete 200 (OK) response. However, a cache MUST NOT store incomplete or partial-content responses if it does not support the Range and Content-Range header fields or if it does not understand the range units used in those fields.

A cache MAY complete a stored incomplete response by making a subsequent range request (Section 9.3 of [Semantics]) and combining the successful response with the stored response, as defined in Section 3.4. A cache MUST NOT use an incomplete response to answer requests unless the response has been made complete or the request is partial and specifies a range that is wholly within the incomplete response. A cache MUST NOT send a partial response to a client without explicitly marking it as such using the 206 (Partial Content) status code.

3.3. Storing Responses to Authenticated Requests

A shared cache MUST NOT use a cached response to a request with an Authorization header field (Section 9.5.3 of [Semantics]) to satisfy any subsequent request unless the response contains a Cache-Control field with a response directive (Section 5.2.2) that allows it to be stored by a shared cache and the cache conforms to the requirements of that directive for that response.

In this specification, the following response directives have such an effect: must-revalidate (Section 5.2.2.1), public (Section 5.2.2.6), and s-maxage (Section 5.2.2.10).

3.4. Combining Partial Content

A response might transfer only a partial representation if the connection closed prematurely or if the request used one or more Range specifiers (Section 9.3 of [Semantics]). After several such transfers, a cache might have received several ranges of the same representation. A cache MAY combine these ranges into a single stored response, and reuse that response to satisfy later requests, if they all share the same strong validator and the cache complies with the client requirements in Section 10.3.7.3 of [Semantics].

When combining the new response with one or more stored responses, a cache MUST use the header fields provided in the new response, aside from Content-Range, to replace all instances of the corresponding header fields in the stored response.

4. Constructing Responses from Caches

When presented with a request, a cache MUST NOT reuse a stored response, unless:

- The presented target URI (Section 6.1 of [Semantics]) and that of the stored response match, and
- the request method associated with the stored response allows it to be used for the presented request, and

- selecting header fields nominated by the stored response (if any) match those presented (see Section 4.1), and

- the stored response does not contain the no-cache cache directive (Section 5.2.2.3), unless it is successfully validated (Section 4.3), and

- the stored response is either:

  - fresh (see Section 4.2), or

  - allowed to be served stale (see Section 4.2.4), or

  - successfully validated (see Section 4.3).

Note that any of the requirements listed above can be overridden by a cache-control extension; see Section 5.2.3.

When a stored response is used to satisfy a request without validation, a cache MUST generate an Age header field (Section 5.1), replacing any present in the response with a value equal to the stored response's current_age; see Section 4.2.3.

A cache MUST write through requests with methods that are unsafe (Section 8.2.1 of [Semantics]) to the origin server; i.e., a cache is not allowed to generate a reply to such a request before having forwarded the request and having received a corresponding response.

Also, note that unsafe requests might invalidate already-stored responses; see Section 4.4.

When more than one suitable response is stored, a cache MUST use the most recent one (as determined by the Date header field). It can also forward the request with "Cache-Control: max-age=0" or "Cache-Control: no-cache" to disambiguate which response to use.

A cache that does not have a clock available MUST NOT use stored responses without revalidating them upon every use.

4.1. Calculating Cache Keys with Vary

When a cache receives a request that can be satisfied by a stored response that has a Vary header field (Section 11.1.4 of [Semantics]), it MUST NOT use that response unless all of the selecting header fields nominated by the Vary header field match in both the original request (i.e., that associated with the stored response), and the presented request.

The selecting header fields from two requests are defined to match if and only if those in the first request can be transformed to those in the second request by applying any of the following:

- adding or removing whitespace, where allowed in the header field's syntax

- combining multiple header fields with the same field name (see Section 5.4 of [Semantics])

- normalizing both header field values in a way that is known to have identical semantics, according to the header field's specification (e.g., reordering field values when order is not significant; case-normalization, where values are defined to be case-insensitive)

If (after any normalization that might take place) a header field is absent from a request, it can only match another request if it is also absent there.

A Vary header field value containing a member "*" always fails to match.

The stored response with matching selecting header fields is known as the selected response.

If multiple selected responses are available (potentially including responses without a Vary header field), the cache will need to choose one to use. When a selecting header field has a known mechanism for doing so (e.g., qvalues on Accept and similar request header fields), that mechanism MAY be used to select preferred responses; of the remainder, the most recent response (as determined by the Date header field) is used, as per Section 4.

Note that in practice, some resources might send the Vary header

field on responses inconsistently. When a cache has multiple responses for a given target URI, and one or more omits the Vary header field, it SHOULD use the most recent non-empty value available to select an appropriate response for the request.

If no selected response is available, the cache cannot satisfy the presented request. Typically, it is forwarded to the origin server in a (possibly conditional; see Section 4.3) request.

4.2. Freshness

A fresh response is one whose age has not yet exceeded its freshness lifetime. Conversely, a stale response is one where it has.

A response's freshness lifetime is the length of time between its generation by the origin server and its expiration time. An explicit expiration time is the time at which the origin server intends that a stored response can no longer be used by a cache without further validation, whereas a heuristic expiration time is assigned by a cache when no explicit expiration time is available.

A response's age is the time that has passed since it was generated by, or successfully validated with, the origin server.

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The primary mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using either the Expires header field (Section 5.3) or the max-age response directive (Section 5.2.2.9). Generally, origin servers will assign future explicit expiration times to responses in the belief that the representation is not likely to change in a semantically significant way before the expiration time is reached.

If an origin server wishes to force a cache to validate every request, it can assign an explicit expiration time in the past to indicate that the response is already stale. Compliant caches will normally validate a stale cached response before reusing it for subsequent requests (see Section 4.2.4).

Since origin servers do not always provide explicit expiration times, caches are also allowed to use a heuristic to determine an expiration time under certain circumstances (see Section 4.2.2).

The calculation to determine if a response is fresh is:

```
response_is_fresh = (freshness_lifetime > current_age)
```

freshness_lifetime is defined in Section 4.2.1; current_age is defined in Section 4.2.3.

Clients can send the max-age or min-fresh request directives (Section 5.2.1) to constrain or relax freshness calculations for the corresponding response. However, caches are not required to honor them.

When calculating freshness, to avoid common problems in date parsing:

- Although all date formats are specified to be case-sensitive, a cache recipient SHOULD match day, week, and time-zone names case-insensitively.
- If a cache recipient's internal implementation of time has less resolution than the value of an HTTP-date, the recipient MUST internally represent a parsed Expires date as the nearest time equal to or earlier than the received value.
- A cache recipient MUST NOT allow local time zones to influence the calculation or comparison of an age or expiration time.

- A cache recipient SHOULD consider a date with a zone abbreviation other than GMT or UTC to be invalid for calculating expiration.

Note that freshness applies only to cache operation; it cannot be used to force a user agent to refresh its display or reload a resource. See Section 6 for an explanation of the difference between caches and history mechanisms.

4.2.1. Calculating Freshness Lifetime

A cache can calculate the freshness lifetime (denoted as

freshness_lifetime) of a response by using the first match of the following:

- If the cache is shared and the s-maxage response directive (Section 5.2.2.10) is present, use its value, or
- If the max-age response directive (Section 5.2.2.9) is present, use its value, or
- If the Expires response header field (Section 5.3) is present, use its value minus the value of the Date response header field, or
- Otherwise, no explicit expiration time is present in the response. A heuristic freshness lifetime might be applicable; see Section 4.2.2.

Note that this calculation is not vulnerable to clock skew, since all of the information comes from the origin server.

When there is more than one value present for a given directive (e.g., two Expires header fields, multiple Cache-Control: max-age directives), the directive's value is considered invalid. Caches are encouraged to consider responses that have invalid freshness information to be stale.

4.2.2. Calculating Heuristic Freshness

Since origin servers do not always provide explicit expiration times, a cache MAY assign a heuristic expiration time when an explicit time is not specified, employing algorithms that use other header field values (such as the Last-Modified time) to estimate a plausible expiration time. This specification does not provide specific algorithms, but does impose worst-case constraints on their results.

A cache MUST NOT use heuristics to determine freshness when an explicit expiration time is present in the stored response. Because of the requirements in Section 3, this means that, effectively,

heuristics can only be used on responses without explicit freshness whose status codes are defined as "heuristically cacheable" (e.g., see Section 10.1 of [Semantics]), and those responses without explicit freshness that have been marked as explicitly cacheable

(e.g., with a "public" response directive).

Note that in previous specifications heuristically cacheable response status codes were called "cacheable by default."

If the response has a Last-Modified header field (Section 11.2.2 of [Semantics]), caches are encouraged to use a heuristic expiration value that is no more than some fraction of the interval since that time. A typical setting of this fraction might be 10%.

> *Note:* Section 13.9 of [RFC2616] prohibited caches from calculating heuristic freshness for URIs with query components (i.e., those containing '?'). In practice, this has not been widely implemented. Therefore, origin servers are encouraged to send explicit directives (e.g., Cache-Control: no-cache) if they wish to preclude caching.

4.2.3. Calculating Age

The Age header field is used to convey an estimated age of the response message when obtained from a cache. The Age field value is the cache's estimate of the number of seconds since the response was generated or validated by the origin server. In essence, the Age value is the sum of the time that the response has been resident in each of the caches along the path from the origin server, plus the amount of time it has been in transit along network paths.

The following data is used for the age calculation:

age_value The term "age_value" denotes the value of the Age header field (Section 5.1), in a form appropriate for arithmetic operation; or 0, if not available.

date_value The term "date_value" denotes the value of the Date header field, in a form appropriate for arithmetic operations. See Section 11.1.1 of [Semantics] for the definition of the Date header field, and for requirements regarding responses without it.

now The term "now" means "the current value of the clock at the host performing the calculation". A host ought to use NTP ([RFC5905]) or some similar protocol to synchronize its clocks to Coordinated Universal Time.

request_time The current value of the clock at the host at the time

the request resulting in the stored response was made.

response_time The current value of the clock at the host at the time the response was received.

A response's age can be calculated in two entirely independent ways:

1. the "apparent_age": response_time minus date_value, if the local clock is reasonably well synchronized to the origin server's clock. If the result is negative, the result is replaced by zero.

2. the "corrected_age_value", if all of the caches along the response path implement HTTP/1.1 or greater. A cache MUST interpret this value relative to the time the request was initiated, not the time that the response was received.

```
apparent_age = max(0, response_time - date_value);

response_delay = response_time - request_time;
corrected_age_value = age_value + response_delay;
```

These are combined as

```
corrected_initial_age = max(apparent_age, corrected_age_value);
```

unless the cache is confident in the value of the Age header field (e.g., because there are no HTTP/1.0 hops in the Via header field), in which case the corrected_age_value MAY be used as the corrected_initial_age.

The current_age of a stored response can then be calculated by adding the amount of time (in seconds) since the stored response was last validated by the origin server to the corrected_initial_age.

```
resident_time = now - response_time;
current_age = corrected_initial_age + resident_time;
```

4.2.4. Serving Stale Responses

A "stale" response is one that either has explicit expiry information or is allowed to have heuristic expiry calculated, but is not fresh according to the calculations in Section 4.2.

A cache MUST NOT generate a stale response if it is prohibited by an explicit in-protocol directive (e.g., by a "no-store" or "no-cache" cache directive, a "must-revalidate" cache-response-directive, or an applicable "s-maxage" or "proxy-revalidate" cache-response-directive; see Section 5.2.2).

A cache MUST NOT generate a stale response unless it is disconnected or doing so is explicitly permitted by the client or origin server (e.g., by the max-stale request directive in Section 5.2.1, by extension directives such as those defined in [RFC5861], or by configuration in accordance with an out-of-band contract).

4.3. Validation

When a cache has one or more stored responses for a requested URI, but cannot serve any of them (e.g., because they are not fresh, or one cannot be selected; see Section 4.1), it can use the conditional request mechanism Section 9.2 of [Semantics] in the forwarded request to give the next inbound server an opportunity to select a valid stored response to use, updating the stored metadata in the process, or to replace the stored response(s) with a new response. This process is known as "validating" or "revalidating" the stored response.

4.3.1. Sending a Validation Request

When generating a conditional request for validation, a cache starts with either a request it is attempting to satisfy, or - if it is initiating the request independently - it synthesises a request using a stored response by copying the method, target URI, and request header fields identified by the Vary header field Section 4.1.

It then updates that request with one or more precondition header fields. These contain validator metadata sourced from stored response(s) that have the same cache key.

The precondition header fields are then compared by recipients to determine whether any stored response is equivalent to a current representation of the resource.

One such validator is the timestamp given in a Last-Modified header field (Section 11.2.2 of [Semantics]), which can be used in an If-Modified-Since header field for response validation, or in an If-Unmodified-Since or If-Range header field for representation selection (i.e., the client is referring specifically to a previously obtained representation with that timestamp).

Another validator is the entity-tag given in an ETag field (Section 11.2.3 of [Semantics]). One or more entity-tags, indicating one or more stored responses, can be used in an If-None-Match header field for response validation, or in an If-Match or If-Range header field for representation selection (i.e., the client is referring specifically to one or more previously obtained representations with the listed entity-tags).

4.3.2. Handling a Received Validation Request

Each client in the request chain may have its own cache, so it is common for a cache at an intermediary to receive conditional requests from other (outbound) caches. Likewise, some user agents make use of conditional requests to limit data transfers to recently modified representations or to complete the transfer of a partially retrieved representation.

If a cache receives a request that can be satisfied by reusing one of its stored 200 (OK) or 206 (Partial Content) responses, the cache SHOULD evaluate any applicable conditional header field preconditions received in that request with respect to the corresponding validators contained within the selected response. A cache MUST NOT evaluate conditional header fields that are only applicable to an origin server, found in a request with semantics that cannot be satisfied with a cached response, or applied to a target resource for which it has no stored responses; such preconditions are likely intended for some other (inbound) server.

The proper evaluation of conditional requests by a cache depends on the received precondition header fields and their precedence, as defined in Section 9.2.2 of [Semantics]. The If-Match and If-Unmodified-Since conditional header fields are not applicable to a cache.

A request containing an If-None-Match header field (Section 9.2.4 of [Semantics]) indicates that the client wants to validate one or more of its own stored responses in comparison to whichever stored response is selected by the cache. If the field value is "*", or if the field value is a list of entity-tags and at least one of them matches the entity-tag of the selected stored response, a cache recipient SHOULD generate a 304 (Not Modified) response (using the metadata of the selected stored response) instead of sending that stored response.

When a cache decides to revalidate its own stored responses for a request that contains an If-None-Match list of entity-tags, the cache MAY combine the received list with a list of entity-tags from its own stored set of responses (fresh or stale) and send the union of the

two lists as a replacement If-None-Match header field value in the forwarded request. If a stored response contains only partial content, the cache MUST NOT include its entity-tag in the union unless the request is for a range that would be fully satisfied by that partial stored response. If the response to the forwarded request is 304 (Not Modified) and has an ETag field value with an entity-tag that is not in the client's list, the cache MUST generate a 200 (OK) response for the client by reusing its corresponding stored response, as updated by the 304 response metadata (Section 4.3.4).

If an If-None-Match header field is not present, a request containing an If-Modified-Since header field (Section 9.2.5 of [Semantics]) indicates that the client wants to validate one or more of its own stored responses by modification date. A cache recipient SHOULD generate a 304 (Not Modified) response (using the metadata of the selected stored response) if one of the following cases is true: 1) the selected stored response has a Last-Modified field value that is earlier than or equal to the conditional timestamp; 2) no Last-Modified field is present in the selected stored response, but it has a Date field value that is earlier than or equal to the conditional timestamp; or, 3) neither Last-Modified nor Date is present in the selected stored response, but the cache recorded it as having been received at a time earlier than or equal to the conditional timestamp.

A cache that implements partial responses to range requests, as defined in Section 9.3 of [Semantics], also needs to evaluate a received If-Range header field (Section 9.2.7 of [Semantics]) with respect to its selected stored response.

4.3.3. Handling a Validation Response

Cache handling of a response to a conditional request is dependent upon its status code:

- A 304 (Not Modified) response status code indicates that the stored response can be updated and reused; see Section 4.3.4.

- A full response (i.e., one with a payload body) indicates that none of the stored responses nominated in the conditional request is suitable. Instead, the cache MUST use the full response to satisfy the request and MAY replace the stored response(s).

- However, if a cache receives a 5xx (Server Error) response while attempting to validate a response, it can either forward this response to the requesting client, or act as if the server failed to respond. In the latter case, the cache MAY send a previously stored response (see Section 4.2.4).

4.3.4. Freshening Stored Responses upon Validation

When a cache receives a 304 (Not Modified) response and already has one or more stored 200 (OK) responses for the applicable cache key, the cache needs to identify which (if any) are to be updated by the new information provided, and then do so.

The stored response(s) to update are identified by using the first match (if any) of the following:

- If the new response contains a strong validator (see Section 11.2.1 of [Semantics]), then that strong validator identifies the selected representation for update. All of the

stored responses with the same strong validator are identified for update. If none of the stored responses contain the same strong validator, then the cache MUST NOT use the new response to update any stored responses.

- If the new response contains a weak validator and that validator corresponds to one of the cache's stored responses, then the most recent of those matching stored responses is identified for update.

- If the new response does not include any form of validator (such as in the case where a client generates an If-Modified-Since request from a source other than the Last-Modified response header field), and there is only one stored response, and that stored response also lacks a validator, then that stored response is identified for update.

For each stored response identified for update, the cache MUST use the header fields provided in the 304 (Not Modified) response to replace all instances of the corresponding header fields in the stored response, with the following exceptions:

- The exceptions to header field storage in Section 3.1 also apply to header field updates.

- Caches MUST NOT update the following header fields: Content-Encoding, Content-Length, Content-MD5 (Section 14.15 of [RFC2616]), Content-Range, ETag.

4.3.5. Freshening Responses with HEAD

A response to the HEAD method is identical to what an equivalent request made with a GET would have been, except it lacks a body. This property of HEAD responses can be used to invalidate or update a cached GET response if the more efficient conditional GET request mechanism is not available (due to no validators being present in the stored response) or if transmission of the representation body is not desired even if it has changed.

When a cache makes an inbound HEAD request for a given target URI and receives a 200 (OK) response, the cache SHOULD update or invalidate

each of its stored GET responses that could have been selected for that request (see Section 4.1).

For each of the stored responses that could have been selected, if the stored response and HEAD response have matching values for any received validator fields (ETag and Last-Modified) and, if the HEAD response has a Content-Length header field, the value of Content-Length matches that of the stored response, the cache SHOULD update the stored response as described below; otherwise, the cache SHOULD consider the stored response to be stale.

If a cache updates a stored response with the metadata provided in a HEAD response, the cache MUST use the header fields provided in the HEAD response to replace all instances of the corresponding header fields in the stored response (subject to the exceptions in Section 4.3.4) and append new header fields to the stored response's header section unless otherwise restricted by the Cache-Control header field.

4.4. Invalidation

Because unsafe request methods (Section 8.2.1 of [Semantics]) such as PUT, POST or DELETE have the potential for changing state on the origin server, intervening caches are required to invalidate stored responses to keep their contents up to date. Invalidate means that the cache will either remove all stored responses whose target URI matches the given URI, or will mark them as "invalid" and in need of a mandatory validation before they can be sent in response to a subsequent request.

Note that this does not guarantee that all appropriate responses are invalidated globally; a state-changing request would only invalidate responses in the caches that it travels through.

A cache MUST invalidate the target URI (Section 6.1 of [Semantics]) as well as the URI(s) in the Location and Content-Location response header fields (if present) when a non-error status code is received in response to an unsafe request method.

However, a cache MUST NOT invalidate a URI from a Location or Content-Location response header field if the host part of that URI differs from the host part in the target URI (Section 6.1 of [Semantics]). This helps prevent denial-of-service attacks.

A cache MUST invalidate the target URI (Section 6.1 of [Semantics]) when it receives a non-error response to a request with a method whose safety is unknown.

Here, a "non-error response" is one with a 2xx (Successful) or 3xx (Redirection) status code.

5. Field Definitions

This section defines the syntax and semantics of HTTP fields related to caching.

Field Name	Status	Reference
Age	standard	Section 5.1
Cache-Control	standard	Section 5.2
Expires	standard	Section 5.3
Pragma	standard	Section 5.4
Warning	obsoleted	Section 5.5

Table 1

5.1. Age

The "Age" header field conveys the sender's estimate of the amount of time since the response was generated or successfully validated at the origin server. Age values are calculated as specified in Section 4.2.3.

```
Age = delta-seconds
```

The Age field value is a non-negative integer, representing time in seconds (see Section 1.3).

The presence of an Age header field implies that the response was not generated or validated by the origin server for this request. However, lack of an Age header field does not imply the origin was contacted, since the response might have been received from an HTTP/1.0 cache that does not implement Age.

5.2. Cache-Control

The "Cache-Control" header field is used to list directives for caches along the request/response chain. Such cache directives are unidirectional in that the presence of a directive in a request does not imply that the same directive is present in the response, or to be repeated in it.

See Section 5.2.3 for information about how Cache-Control directives defined elsewhere are handled.

> | *Note:* Some HTTP/1.0 caches might not implement Cache-Control.

A proxy, whether or not it implements a cache, MUST pass cache directives through in forwarded messages, regardless of their significance to that application, since the directives might be applicable to all recipients along the request/response chain. It is not possible to target a directive to a specific cache.

Cache directives are identified by a token, to be compared case-insensitively, and have an optional argument, that can use both token and quoted-string syntax. For the directives defined below that define arguments, recipients ought to accept both forms, even if a specific form is required for generation.

```
Cache-Control   = 1#cache-directive

cache-directive = token [ "=" ( token / quoted-string ) ]
```

For the cache directives defined below, no argument is defined (nor allowed) unless stated otherwise.

Cache Directive	Reference
max-age	Section 5.2.1.1, Section 5.2.2.9
max-stale	Section 5.2.1.2
min-fresh	Section 5.2.1.3
must-revalidate	Section 5.2.2.1
must-understand	Section 5.2.2.2
no-cache	Section 5.2.1.4, Section 5.2.2.3
no-store	Section 5.2.1.5, Section 5.2.2.4
no-transform	Section 5.2.1.6, Section 5.2.2.5
only-if-cached	Section 5.2.1.7
private	Section 5.2.2.7
proxy-revalidate	Section 5.2.2.8
public	Section 5.2.2.6
s-maxage	Section 5.2.2.10

Table 2

5.2.1. Request Cache-Control Directives

This section defines cache request directives. They are advisory; caches MAY implement them, but are not required to.

5.2.1.1. max-age

Argument syntax:

> delta-seconds (see Section 1.3)

The "max-age" request directive indicates that the client prefers a response whose age is less than or equal to the specified number of seconds. Unless the max-stale request directive is also present, the client does not wish to receive a stale response.

This directive uses the token form of the argument syntax: e.g., 'max-age=5' not 'max-age="5"'. A sender MUST NOT generate the quoted-string form.

5.2.1.2. max-stale

Argument syntax:

delta-seconds (see Section 1.3)

The "max-stale" request directive indicates that the client is willing to accept a response that has exceeded its freshness lifetime. If a value is present, then the client is willing to

accept a response that has exceeded its freshness lifetime by no more than the specified number of seconds. If no value is assigned to max-stale, then the client is willing to accept a stale response of any age.

This directive uses the token form of the argument syntax: e.g., 'max-stale=10' not 'max-stale="10"'. A sender MUST NOT generate the quoted-string form.

5.2.1.3. min-fresh

Argument syntax:

delta-seconds (see Section 1.3)

The "min-fresh" request directive indicates that the client prefers a response whose freshness lifetime is no less than its current age plus the specified time in seconds. That is, the client wants a response that will still be fresh for at least the specified number of seconds.

This directive uses the token form of the argument syntax: e.g., 'min-fresh=20' not 'min-fresh="20"'. A sender MUST NOT generate the quoted-string form.

5.2.1.4. no-cache

The "no-cache" request directive indicates that the client prefers stored response not be used to satisfy the request without successful validation on the origin server.

5.2.1.5. no-store

The "no-store" request directive indicates that a cache MUST NOT store any part of either this request or any response to it. This

directive applies to both private and shared caches. "MUST NOT store" in this context means that the cache MUST NOT intentionally store the information in non-volatile storage, and MUST make a best-effort attempt to remove the information from volatile storage as promptly as possible after forwarding it.

This directive is NOT a reliable or sufficient mechanism for ensuring privacy. In particular, malicious or compromised caches might not recognize or obey this directive, and communications networks might be vulnerable to eavesdropping.

Note that if a request containing this directive is satisfied from a cache, the no-store request directive does not apply to the already stored response.

5.2.1.6. no-transform

The "no-transform" request directive indicates that the client is asking for intermediares (whether or not they implement a cache) to avoid transforming the payload, as defined in Section 6.7.2 of [Semantics].

5.2.1.7. only-if-cached

The "only-if-cached" request directive indicates that the client only wishes to obtain a stored response. Caches that honor this request directive SHOULD, upon receiving it, either respond using a stored response that is consistent with the other constraints of the request, or respond with a 504 (Gateway Timeout) status code.

5.2.2. Response Cache-Control Directives

This section defines cache response directives. A cache MUST obey the requirements of the Cache-Control directives defined in this section.

5.2.2.1. must-revalidate

The "must-revalidate" response directive indicates that once the

response has become stale, a cache MUST NOT reuse that response to satisfy another request until it has been successfully validated by the origin, as defined by Section 4.3.

The must-revalidate directive is necessary to support reliable operation for certain protocol features. In all circumstances a cache MUST obey the must-revalidate directive; in particular, if a cache is disconnected, the cache MUST generate a 504 (Gateway Timeout) response rather than reuse the stale response.

The must-revalidate directive ought to be used by servers if and only if failure to validate a request on the representation could result in incorrect operation, such as a silently unexecuted financial transaction.

The must-revalidate directive also permits a shared cache to reuse a response to a request containing an Authorization header field, subject to the above requirement on revalidation (Section 3.3).

5.2.2.2. must-understand

The "must-understand" response directive limits caching of the response to a cache that understands and conforms to the requirements for that response's status code. A cache MUST NOT store a response containing the must-understand directive if the cache does not understand the response status code.

5.2.2.3. no-cache

Argument syntax:

```
#field-name
```

The "no-cache" response directive, in its unqualified form (without an argument), indicates that the response MUST NOT be used to satisfy any other request without forwarding it for validation and receiving a successful response; see Section 4.3.

This allows an origin server to prevent a cache from using the response to satisfy a request without contacting it, even by caches

that have been configured to send stale responses.

The qualified form of no-cache response directive, with an argument that lists one or more field names, indicates that a cache MAY use the response to satisfy a subsequent request, subject to any other restrictions on caching, if the listed header fields are excluded from the subsequent response or the subsequent response has been successfully revalidated with the origin server (updating or removing those fields). This allows an origin server to prevent the re-use of certain header fields in a response, while still allowing caching of the rest of the response.

The field names given are not limited to the set of header fields defined by this specification. Field names are case-insensitive.

This directive uses the quoted-string form of the argument syntax. A sender SHOULD NOT generate the token form (even if quoting appears not to be needed for single-entry lists).

Note: Although it has been back-ported to many implementations, some HTTP/1.0 caches will not recognize or obey this directive. Also, the qualified form of the directive is often handled by caches as if an unqualified no-cache directive was received; i.e., the special handling for the qualified form is not widely implemented.

5.2.2.4. no-store

The "no-store" response directive indicates that a cache MUST NOT store any part of either the immediate request or response, and MUST NOT use the response to satisfy any other request.

This directive applies to both private and shared caches. "MUST NOT store" in this context means that the cache MUST NOT intentionally store the information in non-volatile storage, and MUST make a best-effort attempt to remove the information from volatile storage as promptly as possible after forwarding it.

This directive is NOT a reliable or sufficient mechanism for ensuring privacy. In particular, malicious or compromised caches might not

recognize or obey this directive, and communications networks might be vulnerable to eavesdropping.

5.2.2.5. no-transform

The "no-transform" response directive indicates that an intermediary (regardless of whether it implements a cache) MUST NOT transform the payload, as defined in Section 6.7.2 of [Semantics].

5.2.2.6. public

The "public" response directive indicates that a cache MAY store the response even if it would otherwise be prohibited, subject to the constraints defined in Section 3. In other words, public explicitly marks the response as cacheable. For example, public permits a shared cache to reuse a response to a request containing an Authorization header field (Section 3.3).

Note that it is not necessary to add the public directive to a response that is already cacheable according to Section 3.

If no explicit freshness information is provided on a response with the public directive, it is heuristically cacheable (Section 4.2.2).

5.2.2.7. private

Argument syntax:

```
#field-name
```


The unqualified "private" response directive indicates that a shared cache MUST NOT store the response (i.e., the response is intended for a single user). It also indicates that a private cache MAY store the response, subject the constraints defined in Section 3, even if the response would not otherwise be heuristically cacheable by a private cache.

If a qualified private response directive is present, with an argument that lists one or more field names, then only the listed fields are limited to a single user: a shared cache MUST NOT store the listed fields if they are present in the original response, but MAY store the remainder of the response message without those fields, subject the constraints defined in Section 3.

The field names given are not limited to the set of header fields defined by this specification. Field names are case-insensitive.

This directive uses the quoted-string form of the argument syntax. A sender SHOULD NOT generate the token form (even if quoting appears not to be needed for single-entry lists).

Note: This usage of the word "private" only controls where the response can be stored; it cannot ensure the privacy of the message content. Also, the qualified form of the directive is often handled by caches as if an unqualified private directive was received; i.e., the special handling for the qualified form is not widely implemented.

5.2.2.8. proxy-revalidate

The "proxy-revalidate" response directive indicates that once the response has become stale, a shared cache MUST NOT reuse that response to satisfy another request until it has been successfully validated by the origin, as defined by Section 4.3. This is analogous to must-revalidate (Section 5.2.2.1), except that proxy-revalidate does not apply to private caches.

Note that "proxy-revalidate" on its own does not imply that a response is cacheable. For example, it might be combined with the public directive (Section 5.2.2.6), allowing the response to be cached while requiring only a shared cache to revalidate when stale.

5.2.2.9. max-age

Argument syntax:

> delta-seconds (see Section 1.3)

The "max-age" response directive indicates that the response is to be considered stale after its age is greater than the specified number of seconds.

This directive uses the token form of the argument syntax: e.g., 'max-age=5' not 'max-age="5"'. A sender MUST NOT generate the quoted-string form.

5.2.2.10. s-maxage

Argument syntax:

> delta-seconds (see Section 1.3)

The "s-maxage" response directive indicates that, for a shared cache, the maximum age specified by this directive overrides the maximum age specified by either the max-age directive or the Expires header field.

The s-maxage directive incorporates the proxy-revalidate (Section 5.2.2.8) response directive's semantics for a shared cache. A shared cache MUST NOT reuse a stale response with s-maxage to satisfy another request until it has been successfully validated by the origin, as defined by Section 4.3. This directive also permits a shared cache to reuse a response to a request containing an Authorization header field, subject to the above requirements on maximum age and revalidation (Section 3.3).

This directive uses the token form of the argument syntax: e.g., 's-maxage=10' not 's-maxage="10"'. A sender MUST NOT generate the quoted-string form.

5.2.3. Cache Control Extensions

The Cache-Control header field can be extended through the use of one or more cache-extension tokens, each with an optional value. A cache MUST ignore unrecognized cache directives.

Informational extensions (those that do not require a change in cache behavior) can be added without changing the semantics of other directives.

Behavioral extensions are designed to work by acting as modifiers to the existing base of cache directives. Both the new directive and the old directive are supplied, such that applications that do not understand the new directive will default to the behavior specified by the old directive, and those that understand the new directive will recognize it as modifying the requirements associated with the old directive. In this way, extensions to the existing cache-control directives can be made without breaking deployed caches.

For example, consider a hypothetical new response directive called "community" that acts as a modifier to the private directive: in addition to private caches, any cache that is shared only by members of the named community is allowed to cache the response. An origin server wishing to allow the UCI community to use an otherwise private response in their shared cache(s) could do so by including

```
Cache-Control: private, community="UCI"
```

A cache that recognizes such a community cache-extension could broaden its behavior in accordance with that extension. A cache that does not recognize the community cache-extension would ignore it and adhere to the private directive.

New extension directives ought to consider defining:

- What it means for a directive to be specified multiple times,
- When the directive does not take an argument, what it means when an argument is present,
- When the directive requires an argument, what it means when it is missing,
- Whether the directive is specific to requests, responses, or able to be used in either.

5.2.4. Cache Directive Registry

The "Hypertext Transfer Protocol (HTTP) Cache Directive Registry" defines the namespace for the cache directives. It has been created and is now maintained at <https://www.iana.org/assignments/http-cache-directives>.

A registration MUST include the following fields:

- Cache Directive Name
- Pointer to specification text

Values to be added to this namespace require IETF Review (see [RFC8126], Section 4.8).

5.3. Expires

The "Expires" header field gives the date/time after which the response is considered stale. See Section 4.2 for further discussion of the freshness model.

The presence of an Expires field does not imply that the original resource will change or cease to exist at, before, or after that time.

The Expires value is an HTTP-date timestamp, as defined in Section 5.4.1.5 of [Semantics].

```
Expires = HTTP-date
```

For example

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

A cache recipient MUST interpret invalid date formats, especially the value "0", as representing a time in the past (i.e., "already expired").

If a response includes a Cache-Control field with the max-age directive (Section 5.2.2.9), a recipient MUST ignore the Expires field. Likewise, if a response includes the s-maxage directive (Section 5.2.2.10), a shared cache recipient MUST ignore the Expires field. In both these cases, the value in Expires is only intended for recipients that have not yet implemented the Cache-Control field.

An origin server without a clock MUST NOT generate an Expires field unless its value represents a fixed time in the past (always expired) or its value has been associated with the resource by a system or

user with a reliable clock.

Historically, HTTP required the Expires field value to be no more than a year in the future. While longer freshness lifetimes are no longer prohibited, extremely large values have been demonstrated to cause problems (e.g., clock overflows due to use of 32-bit integers for time values), and many caches will evict a response far sooner than that.

5.4. Pragma

The "Pragma" header field was defined for HTTP/1.0 caches, so that clients could specify a "no-cache" request (as Cache-Control was not defined until HTTP/1.1).

However, support for Cache-Control is now widespread. As a result, this specification deprecates Pragma.

> | *Note:* Because the meaning of "Pragma: no-cache" in responses was never specified, it does not provide a reliable replacement for "Cache-Control: no-cache" in them.

5.5. Warning

The "Warning" header field was used to carry additional information about the status or transformation of a message that might not be reflected in the status code. This specification obsoletes it, as it is not widely generated or surfaced to users. The information it carried can be gleaned from examining other header fields, such as Age.

6. Relationship to Applications

Applications using HTTP often specify additional forms of caching. For example, Web browsers often have history mechanisms such as "Back" buttons that can be used to redisplay a representation retrieved earlier in a session.

Likewise, some Web browsers implement caching of images and other assets within a page view; they may or may not honor HTTP caching semantics.

The requirements in this specification do not necessarily apply to how applications use data after it is retrieved from a HTTP cache. That is, a history mechanism can display a previous representation even if it has expired, and an application can use cached data in other ways beyond its freshness lifetime.

This does not prohibit the application from taking HTTP caching into account; for example, a history mechanism might tell the user that a view is stale, or it might honor cache directives (e.g., Cache-Control: no-store).

7. Security Considerations

This section is meant to inform developers, information providers, and users of known security concerns specific to HTTP caching. More general security considerations are addressed in HTTP messaging [Messaging] and semantics [Semantics].

Caches expose additional potential vulnerabilities, since the contents of the cache represent an attractive target for malicious exploitation. Because cache contents persist after an HTTP request is complete, an attack on the cache can reveal information long after a user believes that the information has been removed from the network. Therefore, cache contents need to be protected as sensitive information.

7.1. Cache Poisoning

Various attacks might be amplified by being stored in a shared cache. Such "cache poisoning" attacks use the cache to distribute a malicious payload to many clients, and are especially effective when an attacker can use implementation flaws, elevated privileges, or other techniques to insert such a response into a cache.

One common attack vector for cache poisoning is to exploit differences in message parsing on proxies and in user agents; see Section 6.3 of [Messaging] for the relevant requirements regarding HTTP/1.1.

7.2. Timing Attacks

Because one of the primary uses of a cache is to optimise performance, its use can "leak" information about what resources have been previously requested.

For example, if a user visits a site and their browser caches some of its responses, and then navigates to a second site, that site can attempt to load responses that it knows exists on the first site. If they load very quickly, it can be assumed that the user has visited that site, or even a specific page on it.

Such "timing attacks" can be mitigated by adding more information to the cache key, such as the identity of the referring site (to prevent the attack described above). This is sometimes called "double keying."

7.3. Caching of Sensitive Information

Implementation and deployment flaws (as well as misunderstanding of cache operation) might lead to caching of sensitive information (e.g., authentication credentials) that is thought to be private, exposing it to unauthorized parties.

Note that the Set-Cookie response header field [RFC6265] does not inhibit caching; a cacheable response with a Set-Cookie header field can be (and often is) used to satisfy subsequent requests to caches. Servers who wish to control caching of these responses are encouraged to emit appropriate Cache-Control response header fields.

8. IANA Considerations

The change controller for the following registrations is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

8.1. Field Registration

Please update the "Hypertext Transfer Protocol (HTTP) Field Name Registry" at <https://www.iana.org/assignments/http-fields> with the field names listed in the two tables of Section 5.

8.2. Cache Directive Registration

Please update the "Hypertext Transfer Protocol (HTTP) Cache Directive Registry" at <https://www.iana.org/assignments/http-cache-directives> with the registration procedure of Section 5.2.4 and the cache directive names summarized in the table of Section 5.2.

8.3. Warn Code Registry

Please add a note to the "Hypertext Transfer Protocol (HTTP) Warn Codes" registry at <https://www.iana.org/assignments/http-warn-codes> to the effect that Warning is obsoleted.

9. References

9.1. Normative References

[Messaging] Fielding, R., Ed., Nottingham, M., Ed., and J. F. Reschke, Ed., "HTTP/1.1 Messaging", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-10, July 12, 2020, <https://tools.ietf.org/html/draft-ietf-httpbis-messaging-10>.

[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>.

[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/info/rfc3986>.

[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>.

[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., Nottingham, M., Ed., and J. F. Reschke, Ed., "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-10, July 12, 2020, <https://tools.ietf.org/html/draft-ietf-httpbis-semantics-10>.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

9.2. Informative References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/info/rfc2616>.

[RFC5861] Nottingham, M., "HTTP Cache-Control Extensions for Stale Content", RFC 5861, DOI 10.17487/RFC5861, April 2010, <https://www.rfc-editor.org/info/rfc5861>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch,

"Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

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

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

[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/info/rfc8126>.

Appendix A. Collected ABNF

In the collected ABNF below, list rules are expanded as per Section 5.5.1 of [Semantics].

```
Age = delta-seconds

Cache-Control = cache-directive *( OWS "," OWS cache-directive )

Expires = HTTP-date

HTTP-date = <HTTP-date, see [Semantics], Section 5.4.1.5>

OWS = <OWS, see [Semantics], Section 1.2.1>

cache-directive = token [ "=" ( token / quoted-string ) ]

delta-seconds = 1*DIGIT

field-name = <field-name, see [Semantics], Section 5.3>

quoted-string = <quoted-string, see [Semantics], Section 5.4.1.2>

token = <token, see [Semantics], Section 5.4.1.1>
```

Appendix B. Changes from RFC 7234

Some cache directives defined by this specification now have stronger prohibitions against generating the quoted form of their values, since this has been found to create interoperability problems. Consumers of extension cache directives are no longer required to accept both token and quoted-string forms, but they still need to properly parse them for unknown extensions. (Section 5.2)

The "public" and "private" cache directives were clarified, so that they do not make responses reusable under any condition. (Section 5.2.2)

The "must-understand" cache directive was introduced; caches are no longer required to understand the semantics of new response status codes unless it is present. (Section 5.2.2.2)

The Warning response header was obsoleted. Much of the information supported by Warning could be gleaned by examining the response, and the remaining warn-codes - although potentially useful - were entirely advisory. In practice, Warning was not added by caches or intermediaries. (Section 5.5)

Appendix C. Change Log

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

C.1. Between RFC7234 and draft 00

The changes were purely editorial:

- Change boilerplate and abstract to indicate the "draft" status, and update references to ancestor specifications.
- Remove version "1.1" from document title, indicating that this specification applies to all HTTP versions.
- Adjust historical notes.
- Update links to sibling specifications.
- Replace sections listing changes from RFC 2616 by new empty sections referring to RFC 723x.
- Remove acknowledgements specific to RFC 723x.

- Move "Acknowledgements" to the very end and make them unnumbered.

C.2. Since draft-ietf-httpbis-cache-00

The changes are purely editorial:

- Moved all extensibility tips, registration procedures, and registry tables from the IANA considerations to normative sections, reducing the IANA considerations to just instructions that will be removed prior to publication as an RFC.

C.3. Since draft-ietf-httpbis-cache-01

- Cite RFC 8126 instead of RFC 5226 (<https://github.com/httpwg/http-core/issues/75>)

- In Section 5.4, misleading statement about the relation between Pragma and Cache-Control (<https://github.com/httpwg/http-core/issues/92>, <https://www.rfc-editor.org/errata/eid4674>)

C.4. Since draft-ietf-httpbis-cache-02

- In Section 3, explain that only final responses are cacheable (<https://github.com/httpwg/http-core/issues/29>)

- In Section 5.2.2, clarify what responses various directives apply to (<https://github.com/httpwg/http-core/issues/52>)

- In Section 4.3.1, clarify the source of validators in conditional requests (<https://github.com/httpwg/http-core/issues/110>)

- Revise Section 6 to apply to more than just History Lists (<https://github.com/httpwg/http-core/issues/126>)

- In Section 5.5, deprecated "Warning" header field (<https://github.com/httpwg/http-core/issues/139>)

- In Section 3.3, remove a spurious note (<https://github.com/httpwg/http-core/issues/141>)

C.5. Since draft-ietf-httpbis-cache-03

- In Section 2, define what a disconnected cache is (<https://github.com/httpwg/http-core/issues/5>)

- In Section 4, clarify language around how to select a response when more than one matches (<https://github.com/httpwg/http-core/issues/23>)

- in Section 4.2.4, mention stale-while-revalidate and stale-if-error (<https://github.com/httpwg/http-core/issues/122>)

- Remove requirements around cache request directives (<https://github.com/httpwg/http-core/issues/129>)

- Deprecate Pragma (<https://github.com/httpwg/http-core/issues/140>)

- In Section 3.3 and Section 5.2.2, note effect of some directives on authenticated requests (<https://github.com/httpwg/http-core/issues/161>)

C.6. Since draft-ietf-httpbis-cache-04

- In Section 5.2, remove the registrations for stale-if-error and stale-while-revalidate which happened in RFC 7234 (<https://github.com/httpwg/http-core/issues/207>)

C.7. Since draft-ietf-httpbis-cache-05

- In Section 3.2, clarify how weakly framed content is considered for purposes of completeness (<https://github.com/httpwg/http-core/issues/25>)

- Throughout, describe Vary and cache key operations more clearly (<https://github.com/httpwg/http-core/issues/28>)

- In Section 3, remove concept of "cacheable methods" in favor of prose (<https://github.com/httpwg/http-core/issues/54>, <https://www.rfc-editor.org/errata/eid5300>)

- Refactored Section 7, and added a section on timing attacks (<https://github.com/httpwg/http-core/issues/233>)

- Changed "cacheable by default" to "heuristically cacheable" throughout (<https://github.com/httpwg/http-core/issues/242>)

C.8. Since draft-ietf-httpbis-cache-06

- In Section 3 and Section 5.2.2.2, change response cacheability to only require understanding the response status code if the must-understand cache directive is present (<https://github.com/httpwg/http-core/issues/120>)

- Change requirements for handling different forms of cache directives in Section 5.2 (<https://github.com/httpwg/http-core/issues/128>)

- Fix typo in Section 5.2.2.10 (<https://github.com/httpwg/http-core/issues/264>)

- In Section 5.2.2.6 and Section 5.2.2.7, clarify "private" and "public" so that they do not override all other cache directives (<https://github.com/httpwg/http-core/issues/268>)

- In Section 3, distinguish between private with and without qualifying headers (<https://github.com/httpwg/http-core/issues/270>)

- In Section 4.1, clarify that any "*" as a member of Vary will disable caching (<https://github.com/httpwg/http-core/issues/286>)

- In Section 1.1, reference RFC 8174 as well (<https://github.com/httpwg/http-core/issues/303>)

C.9. Since draft-ietf-httpbis-cache-07

- Throughout, replace "effective request URI", "request-target" and similar with "target URI" (<https://github.com/httpwg/http-core/issues/259>)

- In Section 5.2.2.6 and Section 5.2.2.7, make it clear that these directives do not ignore other requirements for caching

(<https://github.com/httpwg/http-core/issues/320>)

- In Section 3.2, move definition of "complete" into semantics (<https://github.com/httpwg/http-core/issues/334>)

C.10. Since draft-ietf-httpbis-cache-08

- Appendix A now uses the sender variant of the "#" list expansion (<https://github.com/httpwg/http-core/issues/192>)

C.11. Since draft-ietf-httpbis-cache-09

- Switch to xml2rfc v3 mode for draft generation (<https://github.com/httpwg/http-core/issues/394>)

Acknowledgments

See Appendix "Acknowledgments" of [Semantics].

Authors' Addresses

Roy T. Fielding (editor)
Adobe
345 Park Ave
San Jose, CA 95110
United States of America

Email: fielding@gbiv.com
URI: https://roy.gbiv.com/

Mark Nottingham (editor)
Fastly

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

Julian F. Reschke (editor)

greenbytes GmbH
Hafenweg 16
48155 Münster
Germany

Email: julian.reschke@greenbytes.de
URI: https://greenbytes.de/tech/webdav/