Network Working Group
Internet-Draft
Obsoletes: 2068, 2616
(if approved)
Intended status: Standards Track
Expires: June 22, 2008

R. Fielding, Ed.
Day Software
J. Gettys
One Laptop per Child
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe Systems
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT

December 20, 2007

HTTP/1.1, part 6: Caching
draft-ietf-httpbis-p6-cache-00

Status of this Memo

Copyright Notice

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 6 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 6 defines requirements on HTTP caches and the associated header fields that control cache behavior or indicate cacheable response messages.

Editorial Note (To be removed by RFC Editor)

This version of the HTTP specification contains only minimal editorial changes from [RFC2616] (abstract, introductory paragraph, and authors' addresses). All other changes are due to partitioning the original into seven mostly independent parts. The intent is for readers of future drafts to able to use draft 00 as the basis for comparison when the WG makes later changes to the specification text. This draft will shortly be followed by draft 01 (containing the first round of changes that have already been agreed to on the mailing list). There is no point in reviewing this draft other than to verify that the partitioning has been done correctly. Roy T. Fielding, Yves Lafon, and Julian Reschke will be the editors after draft 00 is submitted.

Discussion of this draft should take place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org). The current issues list is at <http://www3.tools.ietf.org/wg/httpbis/trac/report/11> and related documents (including fancy diffs) can be found at <http://www3.tools.ietf.org/wg/httpbis/>.

Table of Contents

1. Introduction

This document will define aspects of HTTP related to caching response messages. Right now it only includes the extracted relevant sections of RFC 2616 [RFC2616] without edit.

1.1. Terminology

This specification uses a number of terms to refer to the roles played by participants in, and objects of, the HTTP communication.

cache

> A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. 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 include a cache, though a cache cannot be used by a server that is acting as a tunnel.

cacheable

A response is cacheable if a cache is allowed to store a copy of the response message for use in answering subsequent requests. The rules for determining the cacheability of HTTP responses are defined in Section 2. Even if a resource is cacheable, there may be additional constraints on whether a cache can use the cached copy for a particular request.

first-hand

A response is first-hand if it comes directly and without unnecessary delay from the origin server, perhaps via one or more proxies. A response is also first-hand if its validity has just been checked directly with the origin server.

explicit expiration time

The time at which the origin server intends that an entity should no longer be returned by a cache without further validation.

heuristic expiration time

An expiration time assigned by a cache when no explicit expiration time is available.

age

The age of a response is the time since it was sent by, or successfully validated with, the origin server.

freshness lifetime

The length of time between the generation of a response and its expiration time.

fresh

A response is fresh if its age has not yet exceeded its freshness lifetime.

stale

A response is stale if its age has passed its freshness lifetime.

semantically transparent

A cache behaves in a "semantically transparent" manner, with respect to a particular response, when its use affects neither the requesting client nor the origin server, except to improve performance. When a cache is semantically transparent, the client receives exactly the same response (except for hop-by-hop headers) that it would have received had its request been handled directly by the origin server.

validator

A protocol element (e.g., an entity tag or a Last-Modified time) that is used to find out whether a cache entry is an equivalent copy of an entity.

1.2. Delta Seconds

Some HTTP header fields allow a time value to be specified as an integer number of seconds, represented in decimal, after the time that the message was received.

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

2. Caching in HTTP

2.1. Overview

HTTP is typically used for distributed information systems, where performance can be improved by the use of response caches. The

HTTP/1.1 protocol includes a number of elements intended to make caching work as well as possible. Because these elements are inextricable from other aspects of the protocol, and because they interact with each other, it is useful to describe the basic caching design of HTTP separately from the detailed descriptions of methods, headers, response codes, etc.

Caching would be useless if it did not significantly improve

performance. The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases. The former reduces the number of network round-trips required for many operations; we use an "expiration" mechanism for this purpose (see Section 2.2). The latter reduces network bandwidth requirements; we use a "validation" mechanism for this purpose (see Section 2.3).

Requirements for performance, availability, and disconnected operation require us to be able to relax the goal of semantic transparency. The HTTP/1.1 protocol allows origin servers, caches, and clients to explicitly reduce transparency when necessary. However, because non-transparent operation may confuse non-expert users, and might be incompatible with certain server applications (such as those for ordering merchandise), the protocol requires that transparency be relaxed

- only by an explicit protocol-level request when relaxed by client or origin server

- only with an explicit warning to the end user when relaxed by cache or client

Therefore, the HTTP/1.1 protocol provides these important elements:

1. Protocol features that provide full semantic transparency when this is required by all parties.

2. Protocol features that allow an origin server or user agent to explicitly request and control non-transparent operation.

3. Protocol features that allow a cache to attach warnings to responses that do not preserve the requested approximation of semantic transparency.

A basic principle is that it must be possible for the clients to detect any potential relaxation of semantic transparency.

> Note: The server, cache, or client implementor might be faced with design decisions not explicitly discussed in this specification.

If a decision might affect semantic transparency, the implementor

ought to err on the side of maintaining transparency unless a careful and complete analysis shows significant benefits in breaking transparency.

2.1.1. Cache Correctness

A correct cache MUST respond to a request with the most up-to-date response held by the cache that is appropriate to the request (see sections 2.2.5, 2.2.6, and 2.12) which meets one of the following conditions:

1. It has been checked for equivalence with what the origin server would have returned by revalidating the response with the origin server (Section 2.3);

2. It is "fresh enough" (see Section 2.2). In the default case, this means it meets the least restrictive freshness requirement of the client, origin server, and cache (see Section 3.2); if the origin server so specifies, it is the freshness requirement of the origin server alone. If a stored response is not "fresh enough" by the most restrictive freshness requirement of both the client and the origin server, in carefully considered circumstances the cache MAY still return the response with the appropriate Warning header (see section 2.1.5 and 3.6), unless such a response is prohibited (e.g., by a "no-store" cache-directive, or by a "no-cache" cache-request-directive; see Section 3.2).

3. It is an appropriate 304 (Not Modified), 305 (Proxy Redirect), or error (4xx or 5xx) response message.

If the cache can not communicate with the origin server, then a correct cache SHOULD respond as above if the response can be correctly served from the cache; if not it MUST return an error or warning indicating that there was a communication failure.

If a cache receives a response (either an entire response, or a 304 (Not Modified) response) that it would normally forward to the requesting client, and the received response is no longer fresh, the cache SHOULD forward it to the requesting client without adding a new Warning (but without removing any existing Warning headers). A cache SHOULD NOT attempt to revalidate a response simply because that response became stale in transit; this might lead to an infinite loop. A user agent that receives a stale response without a Warning MAY display a warning indication to the user.

2.1.2. Warnings

Whenever a cache returns a response that is neither first-hand nor "fresh enough" (in the sense of condition 2 in Section 2.1.1), it MUST attach a warning to that effect, using a Warning general-header. The Warning header and the currently defined warnings are described in Section 3.6. The warning allows clients to take appropriate action.

Warnings MAY be used for other purposes, both cache-related and otherwise. The use of a warning, rather than an error status code, distinguish these responses from true failures.

Warnings are assigned three digit warn-codes. The first digit indicates whether the Warning MUST or MUST NOT be deleted from a stored cache entry after a successful revalidation:

1xx Warnings that describe the freshness or revalidation status of the response, and so MUST be deleted after a successful revalidation. 1XX warn-codes MAY be generated by a cache only when validating a cached entry. It MUST NOT be generated by clients.

2xx Warnings that describe some aspect of the entity body or entity headers that is not rectified by a revalidation (for example, a lossy compression of the entity bodies) and which MUST NOT be deleted after a successful revalidation.

See Section 3.6 for the definitions of the codes themselves.

HTTP/1.0 caches will cache all Warnings in responses, without deleting the ones in the first category. Warnings in responses that are passed to HTTP/1.0 caches carry an extra warning-date field, which prevents a future HTTP/1.1 recipient from believing an erroneously cached Warning.

Warnings also carry a warning text. The text MAY be in any appropriate natural language (perhaps based on the client's Accept headers), and include an OPTIONAL indication of what character set is used.

Multiple warnings MAY be attached to a response (either by the origin server or by a cache), including multiple warnings with the same code number. For example, a server might provide the same warning with texts in both English and Basque.

When multiple warnings are attached to a response, it might not be

practical or reasonable to display all of them to the user. This version of HTTP does not specify strict priority rules for deciding

which warnings to display and in what order, but does suggest some heuristics.

2.1.3. Cache-control Mechanisms

The basic cache mechanisms in HTTP/1.1 (server-specified expiration times and validators) are implicit directives to caches. In some cases, a server or client might need to provide explicit directives to the HTTP caches. We use the Cache-Control header for this purpose.

The Cache-Control header allows a client or server to transmit a variety of directives in either requests or responses. These directives typically override the default caching algorithms. As a general rule, if there is any apparent conflict between header values, the most restrictive interpretation is applied (that is, the one that is most likely to preserve semantic transparency). However, in some cases, cache-control directives are explicitly specified as weakening the approximation of semantic transparency (for example, "max-stale" or "public").

The cache-control directives are described in detail in Section 3.2.

2.1.4. Explicit User Agent Warnings

Many user agents make it possible for users to override the basic caching mechanisms. For example, the user agent might allow the user to specify that cached entities (even explicitly stale ones) are never validated. Or the user agent might habitually add "Cache-Control: max-stale=3600" to every request. The user agent SHOULD NOT default to either non-transparent behavior, or behavior that results in abnormally ineffective caching, but MAY be explicitly configured to do so by an explicit action of the user.

If the user has overridden the basic caching mechanisms, the user agent SHOULD explicitly indicate to the user whenever this results in the display of information that might not meet the server's transparency requirements (in particular, if the displayed entity is known to be stale). Since the protocol normally allows the user

agent to determine if responses are stale or not, this indication need only be displayed when this actually happens. The indication need not be a dialog box; it could be an icon (for example, a picture of a rotting fish) or some other indicator.

If the user has overridden the caching mechanisms in a way that would abnormally reduce the effectiveness of caches, the user agent SHOULD continually indicate this state to the user (for example, by a display of a picture of currency in flames) so that the user does not

inadvertently consume excess resources or suffer from excessive latency.

2.1.5. Exceptions to the Rules and Warnings

In some cases, the operator of a cache MAY choose to configure it to return stale responses even when not requested by clients. This decision ought not be made lightly, but may be necessary for reasons of availability or performance, especially when the cache is poorly connected to the origin server. Whenever a cache returns a stale response, it MUST mark it as such (using a Warning header) enabling the client software to alert the user that there might be a potential problem.

It also allows the user agent to take steps to obtain a first-hand or fresh response. For this reason, a cache SHOULD NOT return a stale response if the client explicitly requests a first-hand or fresh one, unless it is impossible to comply for technical or policy reasons.

2.1.6. Client-controlled Behavior

While the origin server (and to a lesser extent, intermediate caches, by their contribution to the age of a response) are the primary source of expiration information, in some cases the client might need to control a cache's decision about whether to return a cached response without validating it. Clients do this using several directives of the Cache-Control header.

A client's request MAY specify the maximum age it is willing to accept of an unvalidated response; specifying a value of zero forces the cache(s) to revalidate all responses. A client MAY also specify the minimum time remaining before a response expires. Both of these

options increase constraints on the behavior of caches, and so cannot further relax the cache's approximation of semantic transparency.

A client MAY also specify that it will accept stale responses, up to some maximum amount of staleness. This loosens the constraints on the caches, and so might violate the origin server's specified constraints on semantic transparency, but might be necessary to support disconnected operation, or high availability in the face of poor connectivity.

2.2. Expiration Model

2.2.1. Server-Specified Expiration

HTTP caching works best when caches can entirely avoid making requests to the origin server. The primary mechanism for avoiding

requests is for an origin server to provide an explicit expiration time in the future, indicating that a response MAY be used to satisfy subsequent requests. In other words, a cache can return a fresh response without first contacting the server.

Our expectation is that servers will assign future explicit expiration times to responses in the belief that the entity is not likely to change, in a semantically significant way, before the expiration time is reached. This normally preserves semantic transparency, as long as the server's expiration times are carefully chosen.

The expiration mechanism applies only to responses taken from a cache and not to first-hand responses forwarded immediately to the requesting client.

If an origin server wishes to force a semantically transparent cache to validate every request, it MAY assign an explicit expiration time in the past. This means that the response is always stale, and so the cache SHOULD validate it before using it for subsequent requests. See Section 3.2.4 for a more restrictive way to force revalidation.

If an origin server wishes to force any HTTP/1.1 cache, no matter how it is configured, to validate every request, it SHOULD use the "must-revalidate" cache-control directive (see Section 3.2).

Servers specify explicit expiration times using either the Expires header, or the max-age directive of the Cache-Control header.

An expiration time cannot be used to force a user agent to refresh its display or reload a resource; its semantics apply only to caching mechanisms, and such mechanisms need only check a resource's expiration status when a new request for that resource is initiated. See Section 2.13 for an explanation of the difference between caches and history mechanisms.

2.2.2. Heuristic Expiration

Since origin servers do not always provide explicit expiration times, HTTP caches typically assign heuristic expiration times, employing algorithms that use other header values (such as the Last-Modified time) to estimate a plausible expiration time. The HTTP/1.1 specification does not provide specific algorithms, but does impose worst-case constraints on their results. Since heuristic expiration times might compromise semantic transparency, they ought to used cautiously, and we encourage origin servers to provide explicit expiration times as much as possible.

2.2.3. Age Calculations

In order to know if a cached entry is fresh, a cache needs to know if its age exceeds its freshness lifetime. We discuss how to calculate the latter in Section 2.2.4; this section describes how to calculate the age of a response or cache entry.

In this discussion, we use the term "now" to mean "the current value of the clock at the host performing the calculation." Hosts that use HTTP, but especially hosts running origin servers and caches, SHOULD use NTP [RFC1305] or some similar protocol to synchronize their clocks to a globally accurate time standard.

HTTP/1.1 requires origin servers to send a Date header, if possible, with every response, giving the time at which the response was generated (see Section 8.3 of [Part1]). We use the term "date_value" to denote the value of the Date header, in a form appropriate for arithmetic operations.

HTTP/1.1 uses the Age response-header to convey the estimated age of the response message when obtained from a cache. The Age field value is the cache's estimate of the amount of time since the response was generated or revalidated 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.

We use the term "age_value" to denote the value of the Age header, in a form appropriate for arithmetic operations.

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

1. now 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. age_value, if all of the caches along the response path implement HTTP/1.1.

Given that we have two independent ways to compute the age of a response when it is received, we can combine these as

    corrected_received_age = max(now - date_value, age_value)

and as long as we have either nearly synchronized clocks or all-HTTP/1.1 paths, one gets a reliable (conservative) result.

Because of network-imposed delays, some significant interval might pass between the time that a server generates a response and the time it is received at the next outbound cache or client. If uncorrected, this delay could result in improperly low ages.

Because the request that resulted in the returned Age value must have been initiated prior to that Age value's generation, we can correct for delays imposed by the network by recording the time at which the request was initiated. Then, when an Age value is received, it MUST be interpreted relative to the time the request was initiated, not the time that the response was received. This algorithm results in

conservative behavior no matter how much delay is experienced. So, we compute:

```
corrected_initial_age = corrected_received_age
                      + (now - request_time)
```

where "request_time" is the time (according to the local clock) when the request that elicited this response was sent.

Summary of age calculation algorithm, when a cache receives a response:

```
/*
 * age_value
 *      is the value of Age: header received by the cache with
 *              this response.
 * date_value
 *      is the value of the origin server's Date: header
 * request_time
 *      is the (local) time when the cache made the request
 *              that resulted in this cached response
 * response_time
 *      is the (local) time when the cache received the
 *              response
 * now
 *      is the current (local) time
 */

apparent_age = max(0, response_time - date_value);
corrected_received_age = max(apparent_age, age_value);
response_delay = response_time - request_time;
corrected_initial_age = corrected_received_age + response_delay;
resident_time = now - response_time;
current_age   = corrected_initial_age + resident_time;
```

The current_age of a cache entry is calculated by adding the amount of time (in seconds) since the cache entry was last validated by the

origin server to the corrected_initial_age. When a response is generated from a cache entry, the cache MUST include a single Age header field in the response with a value equal to the cache entry's current_age.

The presence of an Age header field in a response implies that a response is not first-hand. However, the converse is not true, since the lack of an Age header field in a response does not imply that the response is first-hand unless all caches along the request path are compliant with HTTP/1.1 (i.e., older HTTP caches did not implement the Age header field).

2.2.4. Expiration Calculations

In order to decide whether a response is fresh or stale, we need to compare its freshness lifetime to its age. The age is calculated as described in Section 2.2.3; this section describes how to calculate the freshness lifetime, and to determine if a response has expired. In the discussion below, the values can be represented in any form appropriate for arithmetic operations.

We use the term "expires_value" to denote the value of the Expires header. We use the term "max_age_value" to denote an appropriate value of the number of seconds carried by the "max-age" directive of the Cache-Control header in a response (see Section 3.2.3).

The max-age directive takes priority over Expires, so if max-age is present in a response, the calculation is simply:

```
freshness_lifetime = max_age_value
```

Otherwise, if Expires is present in the response, the calculation is:

```
freshness_lifetime = expires_value - date_value
```

Note that neither of these calculations is vulnerable to clock skew, since all of the information comes from the origin server.

If none of Expires, Cache-Control: max-age, or Cache-Control: s-maxage (see Section 3.2.3) appears in the response, and the response does not include other restrictions on caching, the cache MAY compute a freshness lifetime using a heuristic. The cache MUST attach Warning 113 to any response whose age is more than 24 hours if such warning has not already been added.

Also, if the response does have a Last-Modified time, the heuristic expiration value SHOULD be no more than some fraction of the interval since that time. A typical setting of this fraction might be 10%.

The calculation to determine if a response has expired is quite simple:

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

2.2.5. Disambiguating Expiration Values

Because expiration values are assigned optimistically, it is possible for two caches to contain fresh values for the same resource that are different.

If a client performing a retrieval receives a non-first-hand response for a request that was already fresh in its own cache, and the Date header in its existing cache entry is newer than the Date on the new response, then the client MAY ignore the response. If so, it MAY retry the request with a "Cache-Control: max-age=0" directive (see Section 3.2), to force a check with the origin server.

If a cache has two fresh responses for the same representation with different validators, it MUST use the one with the more recent Date header. This situation might arise because the cache is pooling responses from other caches, or because a client has asked for a reload or a revalidation of an apparently fresh cache entry.

2.2.6. Disambiguating Multiple Responses

Because a client might be receiving responses via multiple paths, so that some responses flow through one set of caches and other responses flow through a different set of caches, a client might receive responses in an order different from that in which the origin server sent them. We would like the client to use the most recently generated response, even if older responses are still apparently fresh.

Neither the entity tag nor the expiration value can impose an ordering on responses, since it is possible that a later response intentionally carries an earlier expiration time. The Date values are ordered to a granularity of one second.

When a client tries to revalidate a cache entry, and the response it receives contains a Date header that appears to be older than the one for the existing entry, then the client SHOULD repeat the request unconditionally, and include

```
Cache-Control: max-age=0
```

to force any intermediate caches to validate their copies directly

with the origin server, or


```
Cache-Control: no-cache
```

to force any intermediate caches to obtain a new copy from the origin server.

If the Date values are equal, then the client MAY use either response (or MAY, if it is being extremely prudent, request a new response). Servers MUST NOT depend on clients being able to choose deterministically between responses generated during the same second, if their expiration times overlap.

2.3. Validation Model

When a cache has a stale entry that it would like to use as a response to a client's request, it first has to check with the origin server (or possibly an intermediate cache with a fresh response) to see if its cached entry is still usable. We call this "validating" the cache entry. Since we do not want to have to pay the overhead of retransmitting the full response if the cached entry is good, and we do not want to pay the overhead of an extra round trip if the cached entry is invalid, the HTTP/1.1 protocol supports the use of conditional methods.

The key protocol features for supporting conditional methods are those concerned with "cache validators." When an origin server generates a full response, it attaches some sort of validator to it, which is kept with the cache entry. When a client (user agent or proxy cache) makes a conditional request for a resource for which it has a cache entry, it includes the associated validator in the request.

The server then checks that validator against the current validator for the entity, and, if they match (see Section 4 of [Part4]), it responds with a special status code (usually, 304 (Not Modified)) and no entity-body. Otherwise, it returns a full response (including entity-body). Thus, we avoid transmitting the full response if the validator matches, and we avoid an extra round trip if it does not match.

In HTTP/1.1, a conditional request looks exactly the same as a normal

request for the same resource, except that it carries a special header (which includes the validator) that implicitly turns the method (usually, GET) into a conditional.

The protocol includes both positive and negative senses of cache-validating conditions. That is, it is possible to request either that a method be performed if and only if a validator matches or if and only if no validators match.

> Note: a response that lacks a validator may still be cached, and served from cache until it expires, unless this is explicitly prohibited by a cache-control directive. However, a cache cannot do a conditional retrieval if it does not have a validator for the entity, which means it will not be refreshable after it expires.

2.3.1. Last-Modified Dates

The Last-Modified entity-header field value is often used as a cache validator. In simple terms, a cache entry is considered to be valid if the entity has not been modified since the Last-Modified value.

2.3.2. Entity Tag Cache Validators

The ETag response-header field value, an entity tag, provides for an "opaque" cache validator. This might allow more reliable validation in situations where it is inconvenient to store modification dates, where the one-second resolution of HTTP date values is not sufficient, or where the origin server wishes to avoid certain paradoxes that might arise from the use of modification dates.

Entity Tags are described in Section 2 of [Part4].

2.3.3. Non-validating Conditionals

The principle behind entity tags is that only the service author knows the semantics of a resource well enough to select an appropriate cache validation mechanism, and the specification of any validator comparison function more complex than byte-equality would open up a can of worms. Thus, comparisons of any other headers (except Last-Modified, for compatibility with HTTP/1.0) are never used for purposes of validating a cache entry.

2.4. Response Cacheability

Unless specifically constrained by a cache-control (Section 3.2) directive, a caching system MAY always store a successful response (see Section 2.8) as a cache entry, MAY return it without validation if it is fresh, and MAY return it after successful validation. If there is neither a cache validator nor an explicit expiration time associated with a response, we do not expect it to be cached, but certain caches MAY violate this expectation (for example, when little or no network connectivity is available). A client can usually detect that such a response was taken from a cache by comparing the Date header to the current time.

> Note: some HTTP/1.0 caches are known to violate this expectation without providing any Warning.

However, in some cases it might be inappropriate for a cache to retain an entity, or to return it in response to a subsequent request. This might be because absolute semantic transparency is deemed necessary by the service author, or because of security or privacy considerations. Certain cache-control directives are therefore provided so that the server can indicate that certain resource entities, or portions thereof, are not to be cached regardless of other considerations.

Note that Section 3.1 of [Part7] normally prevents a shared cache from saving and returning a response to a previous request if that request included an Authorization header.

A response received with a status code of 200, 203, 206, 300, 301 or 410 MAY be stored by a cache and used in reply to a subsequent request, subject to the expiration mechanism, unless a cache-control directive prohibits caching. However, a cache that does not support the Range and Content-Range headers MUST NOT cache 206 (Partial Content) responses.

A response received with any other status code (e.g. status codes 302 and 307) MUST NOT be returned in a reply to a subsequent request unless there are cache-control directives or another header(s) that explicitly allow it. For example, these include the following: an Expires header (Section 3.3); a "max-age", "s-maxage", "must-revalidate", "proxy-revalidate", "public" or "private" cache-control

directive (Section 3.2).

2.5. Constructing Responses From Caches

The purpose of an HTTP cache is to store information received in response to requests for use in responding to future requests. In many cases, a cache simply returns the appropriate parts of a response to the requester. However, if the cache holds a cache entry based on a previous response, it might have to combine parts of a new response with what is held in the cache entry.

2.5.1. End-to-end and Hop-by-hop Headers

For the purpose of defining the behavior of caches and non-caching proxies, we divide HTTP headers into two categories:

- End-to-end headers, which are transmitted to the ultimate recipient of a request or response. End-to-end headers in responses MUST be stored as part of a cache entry and MUST be transmitted in any response formed from a cache entry.

- Hop-by-hop headers, which are meaningful only for a single transport-level connection, and are not stored by caches or forwarded by proxies.

The following HTTP/1.1 headers are hop-by-hop headers:

- Connection
- Keep-Alive
- Proxy-Authenticate
- Proxy-Authorization
- TE
- Trailers
- Transfer-Encoding

- Upgrade

All other headers defined by HTTP/1.1 are end-to-end headers.

Other hop-by-hop headers MUST be listed in a Connection header, (Section 8.1 of [Part1]) to be introduced into HTTP/1.1 (or later).

2.5.2. Non-modifiable Headers

Some features of the HTTP/1.1 protocol, such as Digest Authentication, depend on the value of certain end-to-end headers. A transparent proxy SHOULD NOT modify an end-to-end header unless the definition of that header requires or specifically allows that.

A transparent proxy MUST NOT modify any of the following fields in a request or response, and it MUST NOT add any of these fields if not already present:

- Content-Location
- Content-MD5
- ETag
- Last-Modified

A transparent proxy MUST NOT modify any of the following fields in a response:

- Expires

but it MAY add any of these fields if not already present. If an Expires header is added, it MUST be given a field-value identical to that of the Date header in that response.

A proxy MUST NOT modify or add any of the following fields in a message that contains the no-transform cache-control directive, or in any request:

- Content-Encoding

- Content-Range
- Content-Type

A non-transparent proxy MAY modify or add these fields to a message that does not include no-transform, but if it does so, it MUST add a Warning 214 (Transformation applied) if one does not already appear in the message (see Section 3.6).

> Warning: unnecessary modification of end-to-end headers might cause authentication failures if stronger authentication mechanisms are introduced in later versions of HTTP. Such authentication mechanisms MAY rely on the values of header fields not listed here.

The Content-Length field of a request or response is added or deleted according to the rules in Section 4.4 of [Part1]. A transparent proxy MUST preserve the entity-length (Section 3.2.2 of [Part3]) of the entity-body, although it MAY change the transfer-length (Section 4.4 of [Part1]).

2.5.3. Combining Headers

When a cache makes a validating request to a server, and the server provides a 304 (Not Modified) response or a 206 (Partial Content) response, the cache then constructs a response to send to the requesting client.

If the status code is 304 (Not Modified), the cache uses the entity-body stored in the cache entry as the entity-body of this outgoing response. If the status code is 206 (Partial Content) and the ETag or Last-Modified headers match exactly, the cache MAY combine the contents stored in the cache entry with the new contents received in the response and use the result as the entity-body of this outgoing response, (see Section 4 of [Part5]).

The end-to-end headers stored in the cache entry are used for the constructed response, except that

- any stored Warning headers with warn-code 1xx (see Section 3.6) MUST be deleted from the cache entry and the forwarded response.

- any stored Warning headers with warn-code 2xx MUST be retained in the cache entry and the forwarded response.

- any end-to-end headers provided in the 304 or 206 response MUST replace the corresponding headers from the cache entry.

Unless the cache decides to remove the cache entry, it MUST also replace the end-to-end headers stored with the cache entry with corresponding headers received in the incoming response, except for Warning headers as described immediately above. If a header field-name in the incoming response matches more than one header in the cache entry, all such old headers MUST be replaced.

In other words, the set of end-to-end headers received in the incoming response overrides all corresponding end-to-end headers stored with the cache entry (except for stored Warning headers with warn-code 1xx, which are deleted even if not overridden).

> Note: this rule allows an origin server to use a 304 (Not Modified) or a 206 (Partial Content) response to update any header associated with a previous response for the same entity or sub-ranges thereof, although it might not always be meaningful or correct to do so. This rule does not allow an origin server to use a 304 (Not Modified) or a 206 (Partial Content) response to entirely delete a header that it had provided with a previous response.

2.6. Caching Negotiated Responses

Use of server-driven content negotiation (Section 4.1 of [Part3]), as indicated by the presence of a Vary header field in a response, alters the conditions and procedure by which a cache can use the response for subsequent requests. See Section 3.5 for use of the Vary header field by servers.

A server SHOULD use the Vary header field to inform a cache of what request-header fields were used to select among multiple representations of a cacheable response subject to server-driven negotiation. The set of header fields named by the Vary field value is known as the "selecting" request-headers.

When the cache receives a subsequent request whose Request-URI

specifies one or more cache entries including a Vary header field, the cache MUST NOT use such a cache entry to construct a response to the new request unless all of the selecting request-headers present in the new request match the corresponding stored request-headers in the original request.

The selecting request-headers from two requests are defined to match if and only if the selecting request-headers in the first request can be transformed to the selecting request-headers in the second request by adding or removing linear white space (LWS) at places where this is allowed by the corresponding BNF, and/or combining multiple message-header fields with the same field name following the rules about message headers in Section 4.2 of [Part1].

A Vary header field-value of "*" always fails to match and subsequent requests on that resource can only be properly interpreted by the origin server.

If the selecting request header fields for the cached entry do not match the selecting request header fields of the new request, then the cache MUST NOT use a cached entry to satisfy the request unless it first relays the new request to the origin server in a conditional request and the server responds with 304 (Not Modified), including an entity tag or Content-Location that indicates the entity to be used.

If an entity tag was assigned to a cached representation, the forwarded request SHOULD be conditional and include the entity tags in an If-None-Match header field from all its cache entries for the resource. This conveys to the server the set of entities currently held by the cache, so that if any one of these entities matches the requested entity, the server can use the ETag header field in its 304 (Not Modified) response to tell the cache which entry is appropriate. If the entity-tag of the new response matches that of an existing entry, the new response SHOULD be used to update the header fields of the existing entry, and the result MUST be returned to the client.

If any of the existing cache entries contains only partial content for the associated entity, its entity-tag SHOULD NOT be included in the If-None-Match header field unless the request is for a range that would be fully satisfied by that entry.

If a cache receives a successful response whose Content-Location field matches that of an existing cache entry for the same Request-URI, whose entity-tag differs from that of the existing entry, and whose Date is more recent than that of the existing entry, the existing entry SHOULD NOT be returned in response to future requests and SHOULD be deleted from the cache.

2.7. Shared and Non-Shared Caches

For reasons of security and privacy, it is necessary to make a distinction between "shared" and "non-shared" caches. A non-shared cache is one that is accessible only to a single user. Accessibility in this case SHOULD be enforced by appropriate security mechanisms. All other caches are considered to be "shared." Other sections of this specification place certain constraints on the operation of shared caches in order to prevent loss of privacy or failure of access controls.

2.8. Errors or Incomplete Response Cache Behavior

A cache that receives an incomplete response (for example, with fewer bytes of data than specified in a Content-Length header) MAY store the response. However, the cache MUST treat this as a partial response. Partial responses MAY be combined as described in Section 4 of [Part5]; the result might be a full response or might still be partial. A cache MUST NOT return a partial response to a client without explicitly marking it as such, using the 206 (Partial Content) status code. A cache MUST NOT return a partial response using a status code of 200 (OK).

If a cache receives a 5xx response while attempting to revalidate an entry, it MAY either forward this response to the requesting client, or act as if the server failed to respond. In the latter case, it MAY return a previously received response unless the cached entry includes the "must-revalidate" cache-control directive (see Section 3.2).

2.9. Side Effects of GET and HEAD

Unless the origin server explicitly prohibits the caching of their responses, the application of GET and HEAD methods to any resources SHOULD NOT have side effects that would lead to erroneous behavior if these responses are taken from a cache. They MAY still have side effects, but a cache is not required to consider such side effects in its caching decisions. Caches are always expected to observe an origin server's explicit restrictions on caching.

We note one exception to this rule: since some applications have traditionally used GETs and HEADs with query URLs (those containing a

"?" in the rel_path part) to perform operations with significant side effects, caches MUST NOT treat responses to such URIs as fresh unless the server provides an explicit expiration time. This specifically means that responses from HTTP/1.0 servers for such URIs SHOULD NOT be taken from a cache. See Section 8.1.1 of [Part2] for related information.

2.10. Invalidation After Updates or Deletions

The effect of certain methods performed on a resource at the origin server might cause one or more existing cache entries to become non-transparently invalid. That is, although they might continue to be "fresh," they do not accurately reflect what the origin server would return for a new request on that resource.

There is no way for the HTTP protocol to guarantee that all such cache entries are marked invalid. For example, the request that caused the change at the origin server might not have gone through the proxy where a cache entry is stored. However, several rules help reduce the likelihood of erroneous behavior.

In this section, the phrase "invalidate an entity" means that the cache will either remove all instances of that entity from its storage, or will mark these as "invalid" and in need of a mandatory revalidation before they can be returned in response to a subsequent request.

Some HTTP methods MUST cause a cache to invalidate an entity. This is either the entity referred to by the Request-URI, or by the Location or Content-Location headers (if present). These methods are:

- PUT
- DELETE
- POST

In order to prevent denial of service attacks, an invalidation based on the URI in a Location or Content-Location header MUST only be performed if the host part is the same as in the Request-URI.

A cache that passes through requests for methods it does not understand SHOULD invalidate any entities referred to by the Request-URI.

2.11. Write-Through Mandatory

All methods that might be expected to cause modifications to the origin server's resources MUST be written through to the origin server. This currently includes all methods except for GET and HEAD. A cache MUST NOT reply to such a request from a client before having transmitted the request to the inbound server, and having received a corresponding response from the inbound server. This does not prevent a proxy cache from sending a 100 (Continue) response before

the inbound server has sent its final reply.

The alternative (known as "write-back" or "copy-back" caching) is not allowed in HTTP/1.1, due to the difficulty of providing consistent updates and the problems arising from server, cache, or network failure prior to write-back.

2.12. Cache Replacement

If a new cacheable (see sections 3.2.2, 2.2.5, 2.2.6 and 2.8) response is received from a resource while any existing responses for the same resource are cached, the cache SHOULD use the new response to reply to the current request. It MAY insert it into cache storage and MAY, if it meets all other requirements, use it to respond to any future requests that would previously have caused the old response to be returned. If it inserts the new response into cache storage the rules in Section 2.5.3 apply.

> Note: a new response that has an older Date header value than existing cached responses is not cacheable.

2.13. History Lists

User agents often have history mechanisms, such as "Back" buttons and history lists, which can be used to redisplay an entity retrieved earlier in a session.

History mechanisms and caches are different. In particular history

mechanisms SHOULD NOT try to show a semantically transparent view of the current state of a resource. Rather, a history mechanism is meant to show exactly what the user saw at the time when the resource was retrieved.

By default, an expiration time does not apply to history mechanisms. If the entity is still in storage, a history mechanism SHOULD display it even if the entity has expired, unless the user has specifically configured the agent to refresh expired history documents.

This is not to be construed to prohibit the history mechanism from telling the user that a view might be stale.

> Note: if history list mechanisms unnecessarily prevent users from viewing stale resources, this will tend to force service authors to avoid using HTTP expiration controls and cache controls when they would otherwise like to. Service authors may consider it important that users not be presented with error messages or warning messages when they use navigation controls (such as BACK) to view previously fetched resources. Even though sometimes such

> resources ought not to cached, or ought to expire quickly, user interface considerations may force service authors to resort to other means of preventing caching (e.g. "once-only" URLs) in order not to suffer the effects of improperly functioning history mechanisms.

3. Header Field Definitions

This section defines the syntax and semantics of all standard HTTP/1.1 header fields. For entity-header fields, both sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

3.1. Age

The Age response-header field conveys the sender's estimate of the amount of time since the response (or its revalidation) was generated at the origin server. A cached response is "fresh" if its age does not exceed its freshness lifetime. Age values are calculated as specified in Section 2.2.3.

```
Age = "Age" ":" age-value
age-value = delta-seconds
```

Age values are non-negative decimal integers, representing time in seconds.

If a cache receives a value larger than the largest positive integer it can represent, or if any of its age calculations overflows, it MUST transmit an Age header with a value of 2147483648 (2^31). An HTTP/1.1 server that includes a cache MUST include an Age header field in every response generated from its own cache. Caches SHOULD use an arithmetic type of at least 31 bits of range.

3.2. Cache-Control

The Cache-Control general-header field is used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain. The directives specify behavior intended to prevent caches from adversely interfering with the request or response. These directives typically override the default caching algorithms. Cache directives are unidirectional in that the presence of a directive in a request does not imply that the same directive is to be given in the response.

> Note that HTTP/1.0 caches might not implement Cache-Control and might only implement Pragma: no-cache (see Section 3.4).

Cache directives MUST be passed through by a proxy or gateway application, 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 specify a cache-directive for a specific cache.

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

cache-directive = cache-request-directive
     | cache-response-directive

cache-request-directive =
       "no-cache"                          ; Section 3.2.1
     | "no-store"                          ; Section 3.2.2
```

```
          | "max-age" "=" delta-seconds          ; Section 3.2.3, 3.2.4
          | "max-stale" [ "=" delta-seconds ]    ; Section 3.2.3
          | "min-fresh" "=" delta-seconds        ; Section 3.2.3
          | "no-transform"                       ; Section 3.2.5
          | "only-if-cached"                     ; Section 3.2.4
          | cache-extension                      ; Section 3.2.6

     cache-response-directive =
            "public"                                ; Section 3.2.1
          | "private" [ "=" <"> 1#field-name <"> ]  ; Section 3.2.1
          | "no-cache" [ "=" <"> 1#field-name <"> ]; Section 3.2.1
          | "no-store"                              ; Section 3.2.2
          | "no-transform"                          ; Section 3.2.5
          | "must-revalidate"                       ; Section 3.2.4
          | "proxy-revalidate"                      ; Section 3.2.4
          | "max-age" "=" delta-seconds             ; Section 3.2.3
          | "s-maxage" "=" delta-seconds            ; Section 3.2.3
          | cache-extension                         ; Section 3.2.6

     cache-extension = token [ "=" ( token | quoted-string ) ]
```

When a directive appears without any 1#field-name parameter, the directive applies to the entire request or response. When such a directive appears with a 1#field-name parameter, it applies only to the named field or fields, and not to the rest of the request or response. This mechanism supports extensibility; implementations of future versions of the HTTP protocol might apply these directives to header fields not defined in HTTP/1.1.

The cache-control directives can be broken down into these general categories:

- Restrictions on what are cacheable; these may only be imposed by the origin server.

- Restrictions on what may be stored by a cache; these may be imposed by either the origin server or the user agent.
- Modifications of the basic expiration mechanism; these may be imposed by either the origin server or the user agent.
- Controls over cache revalidation and reload; these may only be

imposed by a user agent.

- Control over transformation of entities.

- Extensions to the caching system.

3.2.1. What is Cacheable

By default, a response is cacheable if the requirements of the request method, request header fields, and the response status indicate that it is cacheable. Section 2.4 summarizes these defaults for cacheability. The following Cache-Control response directives allow an origin server to override the default cacheability of a response:

public

Indicates that the response MAY be cached by any cache, even if it would normally be non-cacheable or cacheable only within a non-shared cache. (See also Authorization, Section 3.1 of [Part7], for additional details.)

private

Indicates that all or part of the response message is intended for a single user and MUST NOT be cached by a shared cache. This allows an origin server to state that the specified parts of the response are intended for only one user and are not a valid response for requests by other users. A private (non-shared) cache MAY cache the response.

Note: This usage of the word private only controls where the response may be cached, and cannot ensure the privacy of the message content.

no-cache

If the no-cache directive does not specify a field-name, then a cache MUST NOT use the response to satisfy a subsequent request without successful revalidation with the origin server. This allows an origin server to prevent caching even by caches that

have been configured to return stale responses to client requests.

If the no-cache directive does specify one or more field-names, then a cache MAY use the response to satisfy a subsequent request, subject to any other restrictions on caching. However, the specified field-name(s) MUST NOT be sent in the response to a subsequent request without successful revalidation with the origin server. 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.

> Note: Most HTTP/1.0 caches will not recognize or obey this directive.

3.2.2. What May be Stored by Caches

no-store

The purpose of the no-store directive is to prevent the inadvertent release or retention of sensitive information (for example, on backup tapes). The no-store directive applies to the entire message, and MAY be sent either in a response or in a request. If sent in a request, a cache MUST NOT store any part of either this request or any response to it. If sent in a response, a cache MUST NOT store any part of either this response or the request that elicited it. This directive applies to both non-shared 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.

Even when this directive is associated with a response, users might explicitly store such a response outside of the caching system (e.g., with a "Save As" dialog). History buffers MAY store such responses as part of their normal operation.

The purpose of this directive is to meet the stated requirements of certain users and service authors who are concerned about accidental releases of information via unanticipated accesses to cache data structures. While the use of this directive might improve privacy in some cases, we caution that it is NOT in any way 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.

3.2.3. Modifications of the Basic Expiration Mechanism

The expiration time of an entity MAY be specified by the origin server using the Expires header (see Section 3.3). Alternatively, it MAY be specified using the max-age directive in a response. When the max-age cache-control directive is present in a cached response, the response is stale if its current age is greater than the age value given (in seconds) at the time of a new request for that resource. The max-age directive on a response implies that the response is cacheable (i.e., "public") unless some other, more restrictive cache directive is also present.

If a response includes both an Expires header and a max-age directive, the max-age directive overrides the Expires header, even if the Expires header is more restrictive. This rule allows an origin server to provide, for a given response, a longer expiration time to an HTTP/1.1 (or later) cache than to an HTTP/1.0 cache. This might be useful if certain HTTP/1.0 caches improperly calculate ages or expiration times, perhaps due to desynchronized clocks.

Many HTTP/1.0 cache implementations will treat an Expires value that is less than or equal to the response Date value as being equivalent to the Cache-Control response directive "no-cache". If an HTTP/1.1 cache receives such a response, and the response does not include a Cache-Control header field, it SHOULD consider the response to be non-cacheable in order to retain compatibility with HTTP/1.0 servers.

> Note: An origin server might wish to use a relatively new HTTP cache control feature, such as the "private" directive, on a network including older caches that do not understand that feature. The origin server will need to combine the new feature with an Expires field whose value is less than or equal to the Date value. This will prevent older caches from improperly caching the response.

s-maxage

> If a response includes an s-maxage directive, then for a shared cache (but not for a private cache), the maximum age specified by this directive overrides the maximum age specified by either the max-age directive or the Expires header. The s-maxage directive also implies the semantics of the proxy-revalidate directive (see Section 3.2.4), i.e., that the shared cache must not use the entry

after it becomes stale to respond to a subsequent request without first revalidating it with the origin server. The s-maxage directive is always ignored by a private cache.

Note that most older caches, not compliant with this specification,

do not implement any cache-control directives. An origin server wishing to use a cache-control directive that restricts, but does not prevent, caching by an HTTP/1.1-compliant cache MAY exploit the requirement that the max-age directive overrides the Expires header, and the fact that pre-HTTP/1.1-compliant caches do not observe the max-age directive.

Other directives allow a user agent to modify the basic expiration mechanism. These directives MAY be specified on a request:

max-age

Indicates that the client is willing to accept a response whose age is no greater than the specified time in seconds. Unless max-stale directive is also included, the client is not willing to accept a stale response.

min-fresh

Indicates that the client is willing to accept 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.

max-stale

Indicates that the client is willing to accept a response that has exceeded its expiration time. If max-stale is assigned a value, then the client is willing to accept a response that has exceeded its expiration time 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.

If a cache returns a stale response, either because of a max-stale directive on a request, or because the cache is configured to

override the expiration time of a response, the cache MUST attach a Warning header to the stale response, using Warning 110 (Response is stale).

A cache MAY be configured to return stale responses without validation, but only if this does not conflict with any "MUST"-level requirements concerning cache validation (e.g., a "must-revalidate" cache-control directive).

If both the new request and the cached entry include "max-age" directives, then the lesser of the two values is used for determining the freshness of the cached entry for that request.

3.2.4. Cache Revalidation and Reload Controls

Sometimes a user agent might want or need to insist that a cache revalidate its cache entry with the origin server (and not just with the next cache along the path to the origin server), or to reload its cache entry from the origin server. End-to-end revalidation might be necessary if either the cache or the origin server has overestimated the expiration time of the cached response. End-to-end reload may be necessary if the cache entry has become corrupted for some reason.

End-to-end revalidation may be requested either when the client does not have its own local cached copy, in which case we call it "unspecified end-to-end revalidation", or when the client does have a local cached copy, in which case we call it "specific end-to-end revalidation."

The client can specify these three kinds of action using Cache-Control request directives:

End-to-end reload

> The request includes a "no-cache" cache-control directive or, for compatibility with HTTP/1.0 clients, "Pragma: no-cache". Field names MUST NOT be included with the no-cache directive in a request. The server MUST NOT use a cached copy when responding to such a request.

Specific end-to-end revalidation

The request includes a "max-age=0" cache-control directive, which forces each cache along the path to the origin server to revalidate its own entry, if any, with the next cache or server. The initial request includes a cache-validating conditional with the client's current validator.

Unspecified end-to-end revalidation

The request includes "max-age=0" cache-control directive, which forces each cache along the path to the origin server to revalidate its own entry, if any, with the next cache or server. The initial request does not include a cache-validating conditional; the first cache along the path (if any) that holds a cache entry for this resource includes a cache-validating conditional with its current validator.

max-age

When an intermediate cache is forced, by means of a max-age=0 directive, to revalidate its own cache entry, and the client has supplied its own validator in the request, the supplied validator might differ from the validator currently stored with the cache entry. In this case, the cache MAY use either validator in making its own request without affecting semantic transparency.

However, the choice of validator might affect performance. The best approach is for the intermediate cache to use its own validator when making its request. If the server replies with 304 (Not Modified), then the cache can return its now validated copy to the client with a 200 (OK) response. If the server replies with a new entity and cache validator, however, the intermediate cache can compare the returned validator with the one provided in the client's request, using the strong comparison function. If the client's validator is equal to the origin server's, then the intermediate cache simply returns 304 (Not Modified). Otherwise, it returns the new entity with a 200 (OK) response.

If a request includes the no-cache directive, it SHOULD NOT include min-fresh, max-stale, or max-age.

only-if-cached

In some cases, such as times of extremely poor network connectivity, a client may want a cache to return only those responses that it currently has stored, and not to reload or revalidate with the origin server. To do this, the client may include the only-if-cached directive in a request. If it receives this directive, a cache SHOULD either respond using a cached entry that is consistent with the other constraints of the request, or respond with a 504 (Gateway Timeout) status. However, if a group of caches is being operated as a unified system with good internal connectivity, such a request MAY be forwarded within that group of caches.

must-revalidate

Because a cache MAY be configured to ignore a server's specified expiration time, and because a client request MAY include a max-stale directive (which has a similar effect), the protocol also includes a mechanism for the origin server to require revalidation of a cache entry on any subsequent use. When the must-revalidate directive is present in a response received by a cache, that cache MUST NOT use the entry after it becomes stale to respond to a subsequent request without first revalidating it with the origin server. (I.e., the cache MUST do an end-to-end revalidation every time, if, based solely on the origin server's Expires or max-age

value, the cached response is stale.)

The must-revalidate directive is necessary to support reliable operation for certain protocol features. In all circumstances an HTTP/1.1 cache MUST obey the must-revalidate directive; in particular, if the cache cannot reach the origin server for any reason, it MUST generate a 504 (Gateway Timeout) response.

Servers SHOULD send the must-revalidate directive if and only if failure to revalidate a request on the entity could result in incorrect operation, such as a silently unexecuted financial transaction. Recipients MUST NOT take any automated action that violates this directive, and MUST NOT automatically provide an unvalidated copy of the entity if revalidation fails.

Although this is not recommended, user agents operating under severe connectivity constraints MAY violate this directive but, if so, MUST explicitly warn the user that an unvalidated response has been provided. The warning MUST be provided on each unvalidated access, and SHOULD require explicit user confirmation.

proxy-revalidate

The proxy-revalidate directive has the same meaning as the must-revalidate directive, except that it does not apply to non-shared user agent caches. It can be used on a response to an authenticated request to permit the user's cache to store and later return the response without needing to revalidate it (since it has already been authenticated once by that user), while still requiring proxies that service many users to revalidate each time (in order to make sure that each user has been authenticated). Note that such authenticated responses also need the public cache control directive in order to allow them to be cached at all.

3.2.5. No-Transform Directive

no-transform

Implementors of intermediate caches (proxies) have found it useful to convert the media type of certain entity bodies. A non-transparent proxy might, for example, convert between image formats in order to save cache space or to reduce the amount of traffic on a slow link.

Serious operational problems occur, however, when these transformations are applied to entity bodies intended for certain kinds of applications. For example, applications for medical imaging, scientific data analysis and those using end-to-end

authentication, all depend on receiving an entity body that is bit for bit identical to the original entity-body.

Therefore, if a message includes the no-transform directive, an intermediate cache or proxy MUST NOT change those headers that are listed in Section 2.5.2 as being subject to the no-transform directive. This implies that the cache or proxy MUST NOT change any aspect of the entity-body that is specified by these headers,

including the value of the entity-body itself.

3.2.6. 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 assigned value. Informational extensions (those which do not require a change in cache behavior) MAY 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 standard directive are supplied, such that applications which do not understand the new directive will default to the behavior specified by the standard directive, and those that understand the new directive will recognize it as modifying the requirements associated with the standard directive. In this way, extensions to the cache-control directives can be made without requiring changes to the base protocol.

This extension mechanism depends on an HTTP cache obeying all of the cache-control directives defined for its native HTTP-version, obeying certain extensions, and ignoring all directives that it does not understand.

For example, consider a hypothetical new response directive called community which acts as a modifier to the private directive. We define this new directive to mean that, in addition to any non-shared cache, any cache which is shared only by members of the community named within its value may 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 seeing this header field will act correctly even if the cache does not understand the community cache-extension, since it will also see and understand the private directive and thus default to the safe behavior.

Unrecognized cache-directives MUST be ignored; it is assumed that any cache-directive likely to be unrecognized by an HTTP/1.1 cache will

be combined with standard directives (or the response's default

cacheability) such that the cache behavior will remain minimally correct even if the cache does not understand the extension(s).

3.3. Expires

The Expires entity-header field gives the date/time after which the response is considered stale. A stale cache entry may not normally be returned by a cache (either a proxy cache or a user agent cache) unless it is first validated with the origin server (or with an intermediate cache that has a fresh copy of the entity). See Section 2.2 for further discussion of the expiration 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 format is an absolute date and time as defined by HTTP-date in Section 3.3.1 of [Part1]; it MUST be in RFC 1123 date format:

```
Expires = "Expires" ":" HTTP-date
```

An example of its use is

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

> Note: if a response includes a Cache-Control field with the max-age directive (see Section 3.2.3), that directive overrides the Expires field.

HTTP/1.1 clients and caches MUST treat other invalid date formats, especially including the value "0", as in the past (i.e., "already expired").

To mark a response as "already expired," an origin server sends an Expires date that is equal to the Date header value. (See the rules for expiration calculations in Section 2.2.4.)

To mark a response as "never expires," an origin server sends an Expires date approximately one year from the time the response is sent. HTTP/1.1 servers SHOULD NOT send Expires dates more than one year in the future.

The presence of an Expires header field with a date value of some time in the future on a response that otherwise would by default be non-cacheable indicates that the response is cacheable, unless indicated otherwise by a Cache-Control header field (Section 3.2).

3.4. Pragma

The Pragma general-header field is used to include implementation-specific directives that might apply to any recipient along the request/response chain. All pragma directives specify optional behavior from the viewpoint of the protocol; however, some systems MAY require that behavior be consistent with the directives.

```
Pragma            = "Pragma" ":" 1#pragma-directive
pragma-directive  = "no-cache" | extension-pragma
extension-pragma  = token [ "=" ( token | quoted-string ) ]
```

When the no-cache directive is present in a request message, an application SHOULD forward the request toward the origin server even if it has a cached copy of what is being requested. This pragma directive has the same semantics as the no-cache cache-directive (see Section 3.2) and is defined here for backward compatibility with HTTP/1.0. Clients SHOULD include both header fields when a no-cache request is sent to a server not known to be HTTP/1.1 compliant.

Pragma directives MUST be passed through by a proxy or gateway application, 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 specify a pragma for a specific recipient; however, any pragma directive not relevant to a recipient SHOULD be ignored by that recipient.

HTTP/1.1 caches SHOULD treat "Pragma: no-cache" as if the client had sent "Cache-Control: no-cache". No new Pragma directives will be defined in HTTP.

> Note: because the meaning of "Pragma: no-cache as a response header field is not actually specified, it does not provide a reliable replacement for "Cache-Control: no-cache" in a response

3.5. Vary

The Vary field value indicates the set of request-header fields that fully determines, while the response is fresh, whether a cache is permitted to use the response to reply to a subsequent request without revalidation. For uncacheable or stale responses, the Vary field value advises the user agent about the criteria that were used to select the representation. A Vary field value of "*" implies that a cache cannot determine from the request headers of a subsequent request whether this response is the appropriate representation. See Section 2.6 for use of the Vary header field by caches.

```
Vary  = "Vary" ":" ( "*" | 1#field-name )
```


An HTTP/1.1 server SHOULD include a Vary header field with any cacheable response that is subject to server-driven negotiation. Doing so allows a cache to properly interpret future requests on that resource and informs the user agent about the presence of negotiation on that resource. A server MAY include a Vary header field with a non-cacheable response that is subject to server-driven negotiation, since this might provide the user agent with useful information about the dimensions over which the response varies at the time of the response.

A Vary field value consisting of a list of field-names signals that the representation selected for the response is based on a selection algorithm which considers ONLY the listed request-header field values in selecting the most appropriate representation. A cache MAY assume that the same selection will be made for future requests with the same values for the listed field names, for the duration of time for which the response is fresh.

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

A Vary field value of "*" signals that unspecified parameters not limited to the request-headers (e.g., the network address of the client), play a role in the selection of the response representation. The "*" value MUST NOT be generated by a proxy server; it may only be generated by an origin server.

3.6. Warning

The Warning general-header field is used to carry additional information about the status or transformation of a message which might not be reflected in the message. This information is typically used to warn about a possible lack of semantic transparency from caching operations or transformations applied to the entity body of the message.

Warning headers are sent with responses using:

```
Warning    = "Warning" ":" 1#warning-value

warning-value = warn-code SP warn-agent SP warn-text
                                          [SP warn-date]

warn-code  = 3DIGIT
warn-agent = ( host [ ":" port ] ) | pseudonym
                ; the name or pseudonym of the server adding
                ; the Warning header, for use in debugging
warn-text  = quoted-string
warn-date  = <"> HTTP-date <">
```

A response MAY carry more than one Warning header.

The warn-text SHOULD be in a natural language and character set that is most likely to be intelligible to the human user receiving the response. This decision MAY be based on any available knowledge, such as the location of the cache or user, the Accept-Language field in a request, the Content-Language field in a response, etc. The default language is English and the default character set is ISO-8859-1.

If a character set other than ISO-8859-1 is used, it MUST be encoded in the warn-text using the method described in RFC 2047 [RFC2047].

Warning headers can in general be applied to any message, however some specific warn-codes are specific to caches and can only be applied to response messages. New Warning headers SHOULD be added after any existing Warning headers. A cache MUST NOT delete any Warning header that it received with a message. However, if a cache successfully validates a cache entry, it SHOULD remove any Warning headers previously attached to that entry except as specified for

specific Warning codes. It MUST then add any Warning headers received in the validating response. In other words, Warning headers are those that would be attached to the most recent relevant response.

When multiple Warning headers are attached to a response, the user agent ought to inform the user of as many of them as possible, in the order that they appear in the response. If it is not possible to inform the user of all of the warnings, the user agent SHOULD follow these heuristics:

- Warnings that appear early in the response take priority over those appearing later in the response.
- Warnings in the user's preferred character set take priority over warnings in other character sets but with identical warn-codes and

warn-agents.

Systems that generate multiple Warning headers SHOULD order them with this user agent behavior in mind.

Requirements for the behavior of caches with respect to Warnings are stated in Section 2.1.2.

This is a list of the currently-defined warn-codes, each with a recommended warn-text in English, and a description of its meaning.

110 Response is stale

MUST be included whenever the returned response is stale.

111 Revalidation failed

MUST be included if a cache returns a stale response because an attempt to revalidate the response failed, due to an inability to reach the server.

112 Disconnected operation

SHOULD be included if the cache is intentionally disconnected from the rest of the network for a period of time.

113 Heuristic expiration

MUST be included if the cache heuristically chose a freshness lifetime greater than 24 hours and the response's age is greater than 24 hours.

199 Miscellaneous warning

The warning text MAY include arbitrary information to be presented to a human user, or logged. A system receiving this warning MUST NOT take any automated action, besides presenting the warning to the user.

214 Transformation applied

MUST be added by an intermediate cache or proxy if it applies any transformation changing the content-coding (as specified in the Content-Encoding header) or media-type (as specified in the Content-Type header) of the response, or the entity-body of the response, unless this Warning code already appears in the response.

299 Miscellaneous persistent warning

The warning text MAY include arbitrary information to be presented to a human user, or logged. A system receiving this warning MUST NOT take any automated action.

If an implementation sends a message with one or more Warning headers whose version is HTTP/1.0 or lower, then the sender MUST include in each warning-value a warn-date that matches the date in the response.

If an implementation receives a message with a warning-value that includes a warn-date, and that warn-date is different from the Date value in the response, then that warning-value MUST be deleted from the message before storing, forwarding, or using it. (This prevents bad consequences of naive caching of Warning header fields.) If all of the warning-values are deleted for this reason, the Warning header MUST be deleted as well.

4. IANA Considerations

TBD.

5. Security Considerations

Caching proxies provide 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 should be protected as sensitive information.

6. Acknowledgments

Much of the content and presentation of the caching design is due to suggestions and comments from individuals including: Shel Kaphan, Paul Leach, Koen Holtman, David Morris, and Larry Masinter.

Based on an XML translation of RFC 2616 by Julian Reschke.

7. References

[Part1] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "HTTP/1.1,

part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-00 (work in progress), December 2007.

[Part2] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "HTTP/1.1, part 2: Message Semantics", draft-ietf-httpbis-p2-semantics-00 (work in progress), December 2007.

[Part3] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H.,

Masinter, L., Leach, P., and T. Berners-Lee, "HTTP/1.1, part 3: Message Payload and Content Negotiation", draft-ietf-httpbis-p3-payload-00 (work in progress), December 2007.

[Part4] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-00 (work in progress), December 2007.

[Part5] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "HTTP/1.1, part 5: Range Requests and Partial Responses", draft-ietf-httpbis-p5-range-00 (work in progress), December 2007.

[Part7] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "HTTP/1.1, part 7: Authentication", draft-ietf-httpbis-p7-auth-00 (work in progress), December 2007.

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation", RFC 1305, March 1992.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

[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, June 1999.

Appendix A. Changes from RFC 2068

A case was missed in the Cache-Control model of HTTP/1.1; s-maxage

was introduced to add this missing case. (Sections 2.4, 3.2, 3.2.3)

The Cache-Control: max-age directive was not properly defined for responses. (Section 3.2.3)

Warnings could be cached incorrectly, or not updated appropriately. (Section 2.1.2, 2.2.4, 2.5.2, 2.5.3, 3.2.3, and 3.6) Warning also needed to be a general header, as PUT or other methods may have need for it in requests.

Index

Authors' Addresses

Roy T. Fielding (editor)
Day Software
23 Corporate Plaza DR, Suite 280
Newport Beach, CA 92660
USA

Phone: +1-949-706-5300
Fax: +1-949-706-5305
Email: fielding@gbiv.com
URI: http://roy.gbiv.com/

Jim Gettys
One Laptop per Child
21 Oak Knoll Road
Carlisle, MA 01741
USA

Email: jg@laptop.org
URI: http://www.laptop.org/

Jeffrey C. Mogul
Hewlett-Packard Company
HP Labs, Large Scale Systems Group
1501 Page Mill Road, MS 1177
Palo Alto, CA 94304
USA

Email: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

Email: henrikn@microsoft.com

Larry Masinter
Adobe Systems, Incorporated
345 Park Ave
San Jose, CA 95110

USA

Email: LMM@acm.org
URI: http://larry.masinter.net/

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

Email: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, Building 32
32 Vassar Street
Cambridge, MA 02139
USA

Email: timbl@w3.org
URI: http://www.w3.org/People/Berners-Lee/

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).