HTTPbis Working Group	R. Fielding, Ed.
Internet-Draft	Day Software
Obsoletes: 2616 (if approved)	J. Gettys
Intended status: Standards Track	Alcatel-Lucent
Expires: February 5, 2011	J. Mogul
	HP
	H. Frystyk
	Microsoft
	L. Masinter
	Adobe Systems
	P. Leach
	Microsoft
	T. Berners-Lee
	W3C/MIT
	Y. Lafon, Ed.
	W3C
	J. Reschke, Ed.
	greenbytes
	August 4, 2010

TOC

HTTP/1.1, part 4: Conditional Requests
draft-ietf-httpbis-p4-conditional-11

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 4 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 4 defines request header fields for indicating conditional requests and the rules for constructing responses to those requests.

Editorial Note (To be removed by RFC Editor)

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://tools.ietf.org/wg/httpbis/trac/report/3 and related documents

(including fancy diffs) can be found at http://tools.ietf.org/wg/httpbis/.
The changes in this draft are summarized in Appendix C.12 (Since draft-ietf-httpbis-p4-conditional-10).

Status of This Memo

Copyright Notice

Table of Contents

---

1. Introduction

TOC

This document defines HTTP/1.1 response metadata for indicating potential changes to payload content, including modification time

stamps and opaque entity-tags, and the HTTP conditional request mechanisms that allow preconditions to be placed on a request method. Conditional GET requests allow for efficient cache updates. Other conditional request methods are used to protect against overwriting or misunderstanding the state of a resource that has been changed unbeknownst to the requesting client.
This document is currently disorganized in order to minimize the changes between drafts and enable reviewers to see the smaller errata changes. The next draft will reorganize the sections to better reflect the content. In particular, the sections on resource metadata will be discussed first and then followed by each conditional request-header, concluding with a definition of precedence and the expectation of ordering strong validator checks before weak validator checks. It is likely that more content from [Part6] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., “HTTP/1.1, part 6: Caching,” August 2010.) will migrate to this part, where appropriate. The current mess reflects how widely dispersed these topics and associated requirements had become in [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.).

1.1. Requirements

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).
An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.2. Syntax Notation

TOC

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” August 2010.) (which extends the syntax defined in [RFC5234]

(Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," January 2008.) with a list rule). Appendix B (Collected ABNF) shows the collected ABNF, with the list rule expanded. The following core rules are included by reference, as defined in [RFC5234] (Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," January 2008.), 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), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), VCHAR (any visible USASCII character), and WSP (whitespace).

1.2.1. Core Rules

TOC

The core rules below are defined in Section 1.2.2 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," August 2010.):

```
quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
OWS           = <OWS, defined in [Part1], Section 1.2.2>
```

1.2.2. ABNF Rules defined in other Parts of the Specification

TOC

The ABNF rules below are defined in other parts:

```
HTTP-date     = <HTTP-date, defined in [Part1], Section 6.1>
```

---

2. Entity-Tags

TOC

Entity-tags are used for comparing two or more representations of the same resource. HTTP/1.1 uses entity-tags in the ETag (Section 6.1 (ETag)), If-Match (Section 6.2 (If-Match)), If-None-Match (Section 6.4 (If-None-Match)), and If-Range (Section 5.3 of [Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses," August 2010.)) header fields. The definition of how they are used and compared as cache validators is in Section 4 (Weak and Strong Validators). An entity-tag consists of an opaque quoted string, possibly prefixed by a weakness indicator.

```
entity-tag = [ weak ] opaque-tag
weak       = %x57.2F ; "W/", case-sensitive
opaque-tag = quoted-string
```

A "strong entity-tag" MAY be shared by two representations of a resource only if they are equivalent by octet equality.
A "weak entity-tag", indicated by the "W/" prefix, MAY be shared by two representations of a resource only if the representations are equivalent and could be substituted for each other with no significant change in semantics. A weak entity-tag can only be used for weak comparison.
An entity-tag MUST be unique across all versions of all representations associated with a particular resource. A given entity-tag value MAY be used for representations obtained by requests on different URIs. The use of the same entity-tag value in conjunction with representations obtained by requests on different URIs does not imply the equivalence of those representations.

2.1. Example: Entity-tags varying on Content-Negotiated Resources

TOC

Consider a resource that is subject to content negotiation (Section 5 of [Part3] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 3: Message Payload and Content Negotiation,” August 2010.)), and where the representations returned upon a GET request vary based on the Accept-Encoding request header field (Section 6.3 of [Part3] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 3: Message Payload and Content Negotiation,” August 2010.)):
>> Request:

```
GET /index HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip
```

In this case, the response might or might not use the gzip content coding. If it does not, the response might look like:
>> Response:

```
HTTP/1.1 200 OK
Date: Thu, 26 Mar 2010 00:05:00 GMT
ETag: "123-a"
Content-Length: 70
Vary: Accept-Encoding
Content-Type: text/plain

Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

An alternative representation that does use gzip content coding would be:
>> Response:

```
HTTP/1.1 200 OK
Date: Thu, 26 Mar 2010 00:05:00 GMT
ETag: "123-b"
Content-Length: 43
Vary: Accept-Encoding
Content-Type: text/plain
Content-Encoding: gzip

...binary data...
```

> Note: Content codings are a property of the representation, so therefore an entity-tag of an encoded representation must be distinct from an unencoded representation to prevent conflicts during cache updates and range requests. In contrast, transfer codings (Section 6.2 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," August 2010.)) apply only during message transfer and do not require distinct entity-tags.

---

3. Status Code Definitions

TOC

TOC

3.1. 304 Not Modified

If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code. The 304 response MUST NOT contain a message-body, and thus is always terminated by the first empty line after the header fields.
A 304 response MUST include a Date header field (Section 9.3 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," August 2010.)) unless its omission is required by Section 9.3.1 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," August 2010.). If a 200 response to the same request would have included any of the header fields Cache-Control, Content-Location, ETag, Expires, Last-Modified, or Vary, then those same header fields MUST be sent in a 304 response. Since the goal of a 304 response is to minimize information transfer when the recipient already has one or more cached representations, the response SHOULD NOT include representation metadata other than the above listed fields unless said metadata exists for the purpose of guiding cache updates (e.g., future HTTP extensions).
If a 304 response includes an entity-tag that indicates a representation not currently cached, then the recipient MUST NOT use the 304 to update its own cache. If that conditional request originated with an outbound client, such as a user agent with its own cache sending a conditional GET to a shared proxy, then the 304 response MAY be forwarded to the outbound client. Otherwise, disregard the response and repeat the request without the conditional.
If a cache uses a received 304 response to update a cache entry, the cache MUST update the entry to reflect any new field values given in the response.

3.2. 412 Precondition Failed

TOC

The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server. This response code allows the client to place preconditions on the current resource metadata (header field data) and thus prevent the requested method from being applied to a resource other than the one intended.

TOC

4. Weak and Strong Validators

Since both origin servers and caches will compare two validators to decide if they represent the same or different representations, one normally would expect that if the representation (including both representation header fields and representation body) changes in any way, then the associated validator would change as well. If this is true, then we call this validator a "strong validator".
However, there might be cases when a server prefers to change the validator only on semantically significant changes, and not when insignificant aspects of the representation change. A validator that does not always change when the representation changes is a "weak validator".
An entity-tag is normally a strong validator, but the protocol provides a mechanism to tag an entity-tag as "weak". One can think of a strong validator as one that changes whenever the sequence of bits in a representation changes, while a weak value changes whenever the meaning of a representation changes. Alternatively, one can think of a strong validator as part of an identifier for a specific representation, whereas a weak validator is part of an identifier for a set of semantically equivalent representations.

> Note: One example of a strong validator is an integer that is incremented in stable storage every time a representation is changed.
>
> A representation's modification time, if defined with only one-second resolution, could be a weak validator, since it is possible that the representation might be modified twice during a single second.
>
> Support for weak validators is optional. However, weak validators allow for more efficient caching of equivalent objects; for example, a hit counter on a site is probably good enough if it is updated every few days or weeks, and any value during that period is likely "good enough" to be equivalent.

A "use" of a validator is either when a client generates a request and includes the validator in a validating header field, or when a server compares two validators.
Strong validators are usable in any context. Weak validators are only usable in contexts that do not depend on exact equality of a representation. For example, either kind is usable for a normal conditional GET. However, only a strong validator is usable for a sub-range retrieval, since otherwise the client might end up with an internally inconsistent representation.
Clients MUST NOT use weak validators in range requests ([Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L.,

Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses," August 2010.).
The only function that HTTP/1.1 defines on validators is comparison. There are two validator comparison functions, depending on whether the comparison context allows the use of weak validators or not:

*The strong comparison function: in order to be considered equal, both opaque-tags MUST be identical character-by-character, and both MUST NOT be weak.

*The weak comparison function: in order to be considered equal, both opaque-tags MUST be identical character-by-character, but either or both of them MAY be tagged as "weak" without affecting the result.

The example below shows the results for a set of entity-tag pairs, and both the weak and strong comparison function results:

ETag 1	ETag 2	Strong Comparison	Weak Comparison
W/"1"	W/"1"	no match	match
W/"1"	W/"2"	no match	no match
W/"1"	"1"	no match	match
"1"	"1"	match	match

An entity-tag is strong unless it is explicitly tagged as weak. Section 2 (Entity-Tags) gives the syntax for entity-tags.
A Last-Modified time, when used as a validator in a request, is implicitly weak unless it is possible to deduce that it is strong, using the following rules:

*The validator is being compared by an origin server to the actual current validator for the representation and,

*That origin server reliably knows that the associated representation did not change twice during the second covered by the presented validator.

or

*The validator is about to be used by a client in an If-Modified-Since or If-Unmodified-Since header, because the client has a cache entry for the associated representation, and

*That cache entry includes a Date value, which gives the time when the origin server sent the original response, and

*The presented Last-Modified time is at least 60 seconds before the Date value.

or

- *The validator is being compared by an intermediate cache to the validator stored in its cache entry for the representation, and

- *That cache entry includes a Date value, which gives the time when the origin server sent the original response, and

- *The presented Last-Modified time is at least 60 seconds before the Date value.

This method relies on the fact that if two different responses were sent by the origin server during the same second, but both had the same Last-Modified time, then at least one of those responses would have a Date value equal to its Last-Modified time. The arbitrary 60-second limit guards against the possibility that the Date and Last-Modified values are generated from different clocks, or at somewhat different times during the preparation of the response. An implementation MAY use a value larger than 60 seconds, if it is believed that 60 seconds is too short.

If a client wishes to perform a sub-range retrieval on a value for which it has only a Last-Modified time and no opaque validator, it MAY do this only if the Last-Modified time is strong in the sense described here.

A cache or origin server receiving a conditional range request ([Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses," August 2010.)) MUST use the strong comparison function to evaluate the condition.

These rules allow HTTP/1.1 caches and clients to safely perform sub-range retrievals on values that have been obtained from HTTP/1.0 servers.

---

5. Rules for When to Use Entity-tags and Last-Modified Dates

TOC

We adopt a set of rules and recommendations for origin servers, clients, and caches regarding when various validator types ought to be used, and for what purposes.

HTTP/1.1 origin servers:

- *SHOULD send an entity-tag validator unless it is not feasible to generate one.

- *MAY send a weak entity-tag instead of a strong entity-tag, if performance considerations support the use of weak entity-tags, or if it is unfeasible to send a strong entity-tag.

*SHOULD send a Last-Modified value if it is feasible to send one, unless the risk of a breakdown in semantic transparency that could result from using this date in an If-Modified-Since header would lead to serious problems.

In other words, the preferred behavior for an HTTP/1.1 origin server is to send both a strong entity-tag and a Last-Modified value.
In order to be legal, a strong entity-tag MUST change whenever the associated representation changes in any way. A weak entity-tag SHOULD change whenever the associated representation changes in a semantically significant way.

> Note: In order to provide semantically transparent caching, an origin server must avoid reusing a specific strong entity-tag value for two different representations, or reusing a specific weak entity-tag value for two semantically different representations. Cache entries might persist for arbitrarily long periods, regardless of expiration times, so it might be inappropriate to expect that a cache will never again attempt to validate an entry using a validator that it obtained at some point in the past.

HTTP/1.1 clients:

*MUST use that entity-tag in any cache-conditional request (using If-Match or If-None-Match) if an entity-tag has been provided by the origin server.

*SHOULD use the Last-Modified value in non-subrange cache-conditional requests (using If-Modified-Since) if only a Last-Modified value has been provided by the origin server.

*MAY use the Last-Modified value in subrange cache-conditional requests (using If-Unmodified-Since) if only a Last-Modified value has been provided by an HTTP/1.0 origin server. The user agent SHOULD provide a way to disable this, in case of difficulty.

*SHOULD use both validators in cache-conditional requests if both an entity-tag and a Last-Modified value have been provided by the origin server. This allows both HTTP/1.0 and HTTP/1.1 caches to respond appropriately.

An HTTP/1.1 origin server, upon receiving a conditional request that includes both a Last-Modified date (e.g., in an If-Modified-Since or If-Unmodified-Since header field) and one or more entity-tags (e.g., in an If-Match, If-None-Match, or If-Range header field) as cache validators, MUST NOT return a response status code of 304 (Not Modified) unless doing so is consistent with all of the conditional header fields in the request.

An HTTP/1.1 caching proxy, upon receiving a conditional request that includes both a Last-Modified date and one or more entity-tags as cache validators, MUST NOT return a locally cached response to the client unless that cached response is consistent with all of the conditional header fields in the request.

> Note: The general principle behind these rules is that HTTP/1.1 servers and clients ought to transmit as much non-redundant information as is available in their responses and requests. HTTP/1.1 systems receiving this information will make the most conservative assumptions about the validators they receive.
>
> HTTP/1.0 clients and caches will ignore entity-tags. Generally, last-modified values received or used by these systems will support transparent and efficient caching, and so HTTP/1.1 origin servers should provide Last-Modified values. In those rare cases where the use of a Last-Modified value as a validator by an HTTP/1.0 system could result in a serious problem, then HTTP/1.1 origin servers should not provide one.

---

6. Header Field Definitions

TOC

This section defines the syntax and semantics of HTTP/1.1 header fields related to conditional requests.

6.1. ETag

TOC

The "ETag" response-header field provides the current value of the entity-tag (see Section 2 (Entity-Tags)) for one representation of the target resource. An entity-tag is intended for use as a resource-local identifier for differentiating between representations of the same resource that vary over time or via content negotiation (see Section 4 (Weak and Strong Validators)).

```
ETag   = "ETag" ":" OWS ETag-v
ETag-v = entity-tag
```

Examples:

```
ETag: "xyzzy"
ETag: W/"xyzzy"
ETag: ""
```

An entity-tag provides an "opaque" cache validator that allows for more reliable validation than modification dates 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.
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.

6.2. If-Match

TOC

The "If-Match" request-header field is used to make a request method conditional. A client that has one or more representations previously obtained from the resource can verify that one of those representations is current by including a list of their associated entity-tags in the If-Match header field.
This allows efficient updates of cached information with a minimum amount of transaction overhead. It is also used when updating resources, to prevent inadvertent modification of the wrong version of a resource. As a special case, the value "*" matches any current representation of the resource.

```
If-Match   = "If-Match" ":" OWS If-Match-v
If-Match-v = "*" / 1#entity-tag
```

If any of the entity-tags match the entity-tag of the representation that would have been returned in the response to a similar GET request (without the If-Match header) on that resource, or if "*" is given and any current representation exists for that resource, then the server MAY perform the requested method as if the If-Match header field did not exist.
If none of the entity-tags match, or if "*" is given and no current representation exists, the server MUST NOT perform the requested method, and MUST return a 412 (Precondition Failed) response. This behavior is most useful when the client wants to prevent an updating method, such as PUT, from modifying a resource that has changed since the client last retrieved it.
If the request would, without the If-Match header field, result in anything other than a 2xx or 412 status code, then the If-Match header MUST be ignored.

The meaning of "If-Match: *" is that the method SHOULD be performed if the representation selected by the origin server (or by a cache, possibly using the Vary mechanism, see Section 3.5 of [Part6] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching," August 2010.)) exists, and MUST NOT be performed if the representation does not exist.
A request intended to update a resource (e.g., a PUT) MAY include an If-Match header field to signal that the request method MUST NOT be applied if the representation corresponding to the If-Match value (a single entity-tag) is no longer a representation of that resource. This allows the user to indicate that they do not wish the request to be successful if the resource has been changed without their knowledge. Examples:

```
If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```

The result of a request having both an If-Match header field and either an If-None-Match or an If-Modified-Since header fields is undefined by this specification.

6.3. If-Modified-Since

TOC

The "If-Modified-Since" request-header field is used to make a request method conditional by date: if the representation that would have been transferred in a 200 response to a GET request has not been modified since the time specified in this field, then do not perform the method; instead, respond as detailed below.

```
If-Modified-Since   = "If-Modified-Since" ":" OWS
                      If-Modified-Since-v
If-Modified-Since-v = HTTP-date
```

An example of the field is:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

A GET method with an If-Modified-Since header and no Range header requests that the representation be transferred only if it has been modified since the date given by the If-Modified-Since header. The algorithm for determining this includes the following cases:

1. If the request would normally result in anything other than a 200 (OK) status code, or if the passed If-Modified-Since date is invalid, the response is exactly the same as for a normal

GET. A date which is later than the server's current time is invalid.

2. If the representation has been modified since the If-Modified-Since date, the response is exactly the same as for a normal GET.

3. If the representation has not been modified since a valid If-Modified-Since date, the server SHOULD return a 304 (Not Modified) response.

The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead.

> Note: The Range request-header field modifies the meaning of If-Modified-Since; see Section 5.4 of [Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses," August 2010.) for full details.
>
> Note: If-Modified-Since times are interpreted by the server, whose clock might not be synchronized with the client.
>
> Note: When handling an If-Modified-Since header field, some servers will use an exact date comparison function, rather than a less-than function, for deciding whether to send a 304 (Not Modified) response. To get best results when sending an If-Modified-Since header field for cache validation, clients are advised to use the exact date string received in a previous Last-Modified header field whenever possible.
>
> Note: If a client uses an arbitrary date in the If-Modified-Since header instead of a date taken from the Last-Modified header for the same request, the client needs to be aware that this date is interpreted in the server's understanding of time. Unsynchronized clocks and rounding problems, due to the different encodings of time between the client and server, are concerns. This includes the possibility of race conditions if the document has changed between the time it was first requested and the If-Modified-Since date of a subsequent request, and the possibility of clock-skew-related problems if the If-Modified-Since date is derived from the client's clock without correction to the server's clock. Corrections for different time bases between client and server are at best approximate due to network latency.

The result of a request having both an If-Modified-Since header field and either an If-Match or an If-Unmodified-Since header fields is undefined by this specification.

6.4. If-None-Match

TOC

The "If-None-Match" request-header field is used to make a request method conditional. A client that has one or more representations previously obtained from the resource can verify that none of those representations is current by including a list of their associated entity-tags in the If-None-Match header field.
This allows efficient updates of cached information with a minimum amount of transaction overhead. It is also used to prevent a method (e.g., PUT) from inadvertently modifying an existing resource when the client believes that the resource does not exist.
As a special case, the value "*" matches any current representation of the resource.

```
If-None-Match   = "If-None-Match" ":" OWS If-None-Match-v
If-None-Match-v = "*" / 1#entity-tag
```

If any of the entity-tags match the entity-tag of the representation that would have been returned in the response to a similar GET request (without the If-None-Match header) on that resource, or if "*" is given and any current representation exists for that resource, then the server MUST NOT perform the requested method, unless required to do so because the resource's modification date fails to match that supplied in an If-Modified-Since header field in the request. Instead, if the request method was GET or HEAD, the server SHOULD respond with a 304 (Not Modified) response, including the cache-related header fields (particularly ETag) of one of the representations that matched. For all other request methods, the server MUST respond with a 412 (Precondition Failed) status code.
If none of the entity-tags match, then the server MAY perform the requested method as if the If-None-Match header field did not exist, but MUST also ignore any If-Modified-Since header field(s) in the request. That is, if no entity-tags match, then the server MUST NOT return a 304 (Not Modified) response.
If the request would, without the If-None-Match header field, result in anything other than a 2xx or 304 status code, then the If-None-Match header MUST be ignored. (See Section 5 (Rules for When to Use Entity-tags and Last-Modified Dates) for a discussion of server behavior when both If-Modified-Since and If-None-Match appear in the same request.)
The meaning of "If-None-Match: *" is that the method MUST NOT be performed if the representation selected by the origin server (or by a cache, possibly using the Vary mechanism, see Section 3.5 of [Part6] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., “HTTP/1.1, part 6: Caching,” August 2010.)) exists, and SHOULD be performed if the representation does not exist. This feature is intended to be useful in preventing races between PUT operations.

Examples:

```
If-None-Match: "xyzzy"
If-None-Match: W/"xyzzy"
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-None-Match: W/"xyzzy", W/"r2d2xxxx", W/"c3piozzzz"
If-None-Match: *
```

The result of a request having both an If-None-Match header field and either an If-Match or an If-Unmodified-Since header fields is undefined by this specification.

6.5. If-Unmodified-Since

TOC

The "If-Unmodified-Since" request-header field is used to make a request method conditional. If the representation that would have been transferred in a 200 response to a GET request on the same resource has not been modified since the time specified in this field, the server SHOULD perform the requested operation as if the If-Unmodified-Since header were not present.
If the representation has been modified since the specified time, the server MUST NOT perform the requested operation, and MUST return a 412 (Precondition Failed).

```
If-Unmodified-Since   = "If-Unmodified-Since" ":" OWS
                        If-Unmodified-Since-v
If-Unmodified-Since-v = HTTP-date
```

An example of the field is:

```
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

If the request normally (i.e., without the If-Unmodified-Since header) would result in anything other than a 2xx or 412 status code, the If-Unmodified-Since header SHOULD be ignored.
If the specified date is invalid, the header is ignored.
The result of a request having both an If-Unmodified-Since header field and either an If-None-Match or an If-Modified-Since header fields is undefined by this specification.

6.6. Last-Modified

TOC

The "Last-Modified" header field indicates the date and time at which the origin server believes the representation was last modified.

```
Last-Modified   = "Last-Modified" ":" OWS Last-Modified-v
Last-Modified-v = HTTP-date
```

An example of its use is

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

The exact meaning of this header field depends on the implementation of the origin server and the nature of the original resource. For files, it might be just the file system last-modified time. For representations with dynamically included parts, it might be the most recent of the set of last-modify times for its component parts. For database gateways, it might be the last-update time stamp of the record. For virtual objects, it might be the last time the internal state changed.

An origin server MUST NOT send a Last-Modified date which is later than the server's time of message origination. In such cases, where the resource's last modification would indicate some time in the future, the server MUST replace that date with the message origination date.

An origin server SHOULD obtain the Last-Modified value of the representation as close as possible to the time that it generates the Date value of its response. This allows a recipient to make an accurate assessment of the representation's modification time, especially if the representation changes near the time that the response is generated.

HTTP/1.1 servers SHOULD send Last-Modified whenever feasible.

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

7. IANA Considerations

TOC

7.1. Status Code Registration

TOC

The HTTP Status Code Registry located at http://www.iana.org/assignments/http-status-codes shall be updated with the registrations below:

Value	Description	Reference
304	Not Modified	Section 3.1 (304 Not Modified)

412	Precondition Failed	Section 3.2 (412 Precondition Failed)

7.2. Header Field Registration

TOC

The Message Header Field Registry located at http://www.iana.org/assignments/message-headers/message-header-index.html shall be updated with the permanent registrations below (see [RFC3864] (Klyne, G., Nottingham, M., and J. Mogul, “Registration Procedures for Message Header Fields,” September 2004.)):

Header Field Name	Protocol	Status	Reference
ETag	http	standard	Section 6.1 (ETag)
If-Match	http	standard	Section 6.2 (If-Match)
If-Modified-Since	http	standard	Section 6.3 (If-Modified-Since)
If-None-Match	http	standard	Section 6.4 (If-None-Match)
If-Unmodified-Since	http	standard	Section 6.5 (If-Unmodified-Since)
Last-Modified	http	standard	Section 6.6 (Last-Modified)

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

---

8. Security Considerations

TOC

No additional security considerations have been identified beyond those applicable to HTTP in general [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” August 2010.).

---

9. Acknowledgments

TOC

---

10. References

TOC

10.1. Normative References

TOC

[Part1]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," draft-ietf-httpbis-p1-messaging-11 (work in progress), August 2010.
[Part3]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 3: Message Payload and Content Negotiation," draft-ietf-httpbis-p3-payload-11 (work in progress), August 2010.
[Part5]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses," draft-ietf-httpbis-p5-range-11 (work in progress), August 2010.
[Part6]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching," draft-ietf-httpbis-p6-cache-11 (work in progress), August 2010.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997.
[RFC5234]	Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," STD 68, RFC 5234, January 2008.

10.2. Informative References

TOC

[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.
[RFC3864]	Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields," BCP 90, RFC 3864, September 2004.

Appendix A. Changes from RFC 2616

TOC

Allow weak entity-tags in all requests except range requests (Sections 4 (Weak and Strong Validators) and 6.4 (If-None-Match)).

Appendix B. Collected ABNF

TOC

```
     ETag = "ETag:" OWS ETag-v
     ETag-v = entity-tag

     HTTP-date = <HTTP-date, defined in [Part1], Section 6.1>

     If-Match = "If-Match:" OWS If-Match-v
     If-Match-v = "*" / ( *( "," OWS ) entity-tag *( OWS "," [ OWS
      entity-tag ] ) )
     If-Modified-Since = "If-Modified-Since:" OWS If-Modified-Since-v
     If-Modified-Since-v = HTTP-date
     If-None-Match = "If-None-Match:" OWS If-None-Match-v
     If-None-Match-v = "*" / ( *( "," OWS ) entity-tag *( OWS "," [ OWS
      entity-tag ] ) )
     If-Unmodified-Since = "If-Unmodified-Since:" OWS
      If-Unmodified-Since-v
     If-Unmodified-Since-v = HTTP-date

     Last-Modified = "Last-Modified:" OWS Last-Modified-v
     Last-Modified-v = HTTP-date

     OWS = <OWS, defined in [Part1], Section 1.2.2>

     entity-tag = [ weak ] opaque-tag

     opaque-tag = quoted-string

     quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>

     weak = %x57.2F ; W/

ABNF diagnostics:

     ; ETag defined but not used
     ; If-Match defined but not used
     ; If-Modified-Since defined but not used
     ; If-None-Match defined but not used
     ; If-Unmodified-Since defined but not used
     ; Last-Modified defined but not used
```

Appendix C. Change Log (to be removed by RFC Editor before publication)

TOC

C.1. Since RFC2616

TOC

Extracted relevant partitions from [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.).

C.2. Since draft-ietf-httpbis-p4-conditional-00

TOC

Closed issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/35: "Normative and Informative references"

Other changes:

- Move definitions of 304 and 412 condition codes from Part2.

C.3. Since draft-ietf-httpbis-p4-conditional-01

TOC

Ongoing work on ABNF conversion (http://tools.ietf.org/wg/httpbis/trac/ticket/36):

- Add explicit references to BNF syntax and rules imported from other parts of the specification.

C.4. Since draft-ietf-httpbis-p4-conditional-02

TOC

Closed issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/116: "Weak ETags on non-GET requests"

Ongoing work on IANA Message Header Registration (http://tools.ietf.org/wg/httpbis/trac/ticket/40):

- Reference RFC 3984, and update header registrations for headers defined in this document.

C.5. Since draft-ietf-httpbis-p4-conditional-03

TOC

Closed issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/71: "Examples for ETag matching"
- http://tools.ietf.org/wg/httpbis/trac/ticket/124: "'entity value' undefined"
- http://tools.ietf.org/wg/httpbis/trac/ticket/126: "bogus 2068 Date header reference"

C.6. Since draft-ietf-httpbis-p4-conditional-04

TOC

Ongoing work on ABNF conversion (http://tools.ietf.org/wg/httpbis/trac/ticket/36):

- Use "/" instead of "|" for alternatives.
- Introduce new ABNF rules for "bad" whitespace ("BWS"), optional whitespace ("OWS") and required whitespace ("RWS").
- Rewrite ABNFs to spell out whitespace rules, factor out header value format definitions.

C.7. Since draft-ietf-httpbis-p4-conditional-05

TOC

Final work on ABNF conversion (http://tools.ietf.org/wg/httpbis/trac/ticket/36):

- Add appendix containing collected and expanded ABNF, reorganize ABNF introduction.

TOC

C.8. Since draft-ietf-httpbis-p4-conditional-06

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/153: "case-sensitivity of etag weakness indicator"

C.9. Since draft-ietf-httpbis-p4-conditional-07 TOC

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/116: "Weak ETags on non-GET requests" (If-Match still was defined to require strong matching)

* http://tools.ietf.org/wg/httpbis/trac/ticket/198: "move IANA registrations for optional status codes"

C.10. Since draft-ietf-httpbis-p4-conditional-08 TOC

No significant changes.

C.11. Since draft-ietf-httpbis-p4-conditional-09 TOC

No significant changes.

C.12. Since draft-ietf-httpbis-p4-conditional-10 TOC

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/69: "Clarify 'Requested Variant'"

* http://tools.ietf.org/wg/httpbis/trac/ticket/109: "Clarify entity / representation / variant terminology"

* http://tools.ietf.org/wg/httpbis/trac/ticket/220: "consider removing the 'changes from 2068' sections"

Index

TOC

	Status Codes
	304 Not Modified
	412 Precondition Failed

Authors' Addresses

TOC

	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
	Alcatel-Lucent Bell Labs
	21 Oak Knoll Road
	Carlisle, MA 01741
	USA
EMail:	jg@freedesktop.org
URI:	http://gettys.wordpress.com/
	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/
	Yves Lafon (editor)
	World Wide Web Consortium
	W3C / ERCIM
	2004, rte des Lucioles
	Sophia-Antipolis, AM 06902
	France
EMail:	ylafon@w3.org
URI:	http://www.raubacapeu.net/people/yves/
	Julian F. Reschke (editor)
	greenbytes GmbH
	Hafenweg 16
	Muenster, NW 48155
	Germany
Phone:	+49 251 2807760
Fax:	+49 251 2807761
EMail:	julian.reschke@greenbytes.de
URI:	http://greenbytes.de/tech/webdav/