HTTPbis Working Group	R. Fielding, Ed.
Internet-Draft	Adobe
Obsoletes: 2616 (if approved)	J. Gettys
Intended status: Standards Track	Alcatel-Lucent
Expires: October 20, 2011	J. Mogul
	HP
	H. Frystyk
	Microsoft
	L. Masinter
	Adobe
	P. Leach
	Microsoft
	T. Berners-Lee
	W3C/MIT
	Y. Lafon, Ed.
	W3C
	J. F. Reschke, Ed.
	greenbytes
	April 18, 2011

HTTP/1.1, part 4: Conditional Requests
draft-ietf-httpbis-p4-conditional-14

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), which is archived at http://lists.w3.org/Archives/Public/ietf-http-wg/.
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 Appendix C.15.

Status of this Memo

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

Copyright Notice

Table of Contents

1. Introduction

This document defines the HTTP/1.1 conditional request mechanisms, including both response metadata that can be used to indicate or observe changes to resource state and request header fields that specify preconditions to be checked before performing the action given by the request method. Conditional GET requests are the most efficient mechanism for HTTP cache updates [Part6]. Conditionals can also be applied to state-changing methods, such as PUT and DELETE, to prevent the "lost update" problem: one client accidentally overwriting the work of another client that has been acting in parallel.
Conditional request preconditions are based on the state of the target resource as a whole (its current value set) or the state as observed in a previously obtained representation (one value in that set). A resource might have multiple current representations, each with its own observable state. The conditional request mechanisms assume that the

mapping of requests to corresponding representations will be consistent over time if the server intends to take advantage of conditionals. Regardless, if the mapping is inconsistent and the server is unable to select the appropriate representation, then no harm will result when the precondition evaluates to false.

We use the term "selected representation" to refer to the current representation of the target resource that would have been selected in a successful response if the same request had used the method GET and had excluded all of the conditional request header fields. The conditional request preconditions are evaluated by comparing the values provided in the request header fields to the current metadata for the selected representation.

1.1. Requirements

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].

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

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). Appendix Appendix B shows the collected ABNF, with the list rule expanded.

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

The ABNF rules below are defined in other parts:

```
  quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
  OWS           = <OWS, defined in [Part1], Section 1.2.2>
  HTTP-date     = <HTTP-date, defined in [Part1], Section 6.1>
```

2. Resource State Metadata (Validators)

This specification defines two forms of metadata that are commonly used to observe resource state and test for preconditions: modification dates and opaque entity tags. Additional metadata that reflects

resource state has been defined by various extensions of HTTP, such as WebDAV [RFC4918], that are beyond the scope of this specification. A resource metadata value is referred to as a "validator" when it is used within a precondition.

2.1. Last-Modified

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

```
  Last-Modified = HTTP-date
```

An example of its use is

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

2.1.1. Generation

Origin servers SHOULD send Last-Modified for any selected representation for which a last modification date can be reasonably and consistently determined, since its use in conditional requests and evaluating cache freshness ([Part6]) results in a substantial reduction of HTTP traffic on the Internet and can be a significant factor in improving service scalability and reliability.

A representation is typically the sum of many parts behind the resource interface. The last-modified time would usually be the most recent time that any of those parts were changed. How that value is determined for any given resource is an implementation detail beyond the scope of this specification. What matters to HTTP is how recipients of the Last-Modified header field can use its value to make conditional requests and test the validity of locally cached responses.

An origin server SHOULD obtain the Last-Modified value of the representation as close as possible to the time that it generates the Date field-value for 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.

An origin server with a clock MUST NOT send a Last-Modified date that is later than the server's time of message origination (Date). If the last modification time is derived from implementation-specific metadata that evaluates to some time in the future, according to the origin server's clock, then the origin server MUST replace that value with the message origination date. This prevents a future modification date from having an adverse impact on cache validation.

2.1.2. Comparison

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 field, 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.

2.2. ETag

The ETag header field provides the current entity-tag for the selected representation. An entity-tag is an opaque validator for

differentiating between multiple representations of the same resource, regardless of whether those multiple representations are due to resource state changes over time, content negotiation resulting in multiple representations being valid at the same time, or both. An entity-tag consists of an opaque quoted string, possibly prefixed by a weakness indicator.

```
  ETag       = entity-tag

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

An entity-tag can be more reliable for validation than a modification date in situations where it is inconvenient to store modification dates, where the one-second resolution of HTTP date values is not sufficient, or where modification dates are not consistently maintained.
Examples:

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

2.2.1. Generation

The principle behind entity-tags is that only the service author knows the implementation of a resource well enough to select the most accurate and efficient validation mechanism for that resource, and that any such mechanism can be mapped to a simple sequence of octets for easy comparison. Since the value is opaque, there is no need for the client to be aware of how each entity-tag is constructed.
For example, a resource that has implementation-specific versioning applied to all changes might use an internal revision number, perhaps combined with a variance identifier for content negotiation, to accurately differentiate between representations. Other implementations might use a stored hash of representation content, a combination of various filesystem attributes, or a modification timestamp that has sub-second resolution.
Origin servers SHOULD send ETag for any selected representation for which detection of changes can be reasonably and consistently determined, since the entity-tag's use in conditional requests and evaluating cache freshness ([Part6]) can result in a substantial reduction of HTTP network traffic and can be a significant factor in improving service scalability and reliability.

2.2.2. Weak versus Strong

Since both origin servers and caches will compare two validators to decide if they indicate 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 that validator a "strong validator". One example of a strong validator is an integer that is incremented in stable storage every time a representation is changed.
However, there might be cases when a server prefers to change the validator only when it desires cached representations to be invalidated. For example, the representation of a weather report that changes in content every second, based on dynamic measurements, might be grouped into sets of equivalent representations (from the origin server's perspective) in order to allow cached representations to be valid for a reasonable period of time (perhaps adjusted dynamically based on server load or weather quality). A validator that does not always change when the representation changes is a "weak validator".
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 equivalent representations (where this notion of equivalence is entirely governed by the origin server and beyond the scope of this specification).
An entity-tag is normally a strong validator, but the protocol provides a mechanism to tag an entity-tag as "weak".

- 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 strong entity-tag MUST change whenever the associated representation changes in any way. A weak entity-tag SHOULD change whenever the origin server considers prior representations to be unacceptable as a substitute for the current representation. In other words, a weak entity tag SHOULD change whenever the origin server wants caches to invalidate old responses.
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. A weak entity-tag can only be used for weak comparison.
Cache entries might persist for arbitrarily long periods, regardless of expiration times. Thus, a cache might attempt to validate an entry using a validator that it obtained in the distant past. A strong entity-tag MUST be unique across all versions of all representations

associated with a particular resource over time. However, there is no implication of uniqueness across entity-tags of different resources (i.e., the same entity-tag value might be in use for representations of multiple resources at the same time and does not imply that those representations are equivalent).

2.2.3. Comparison

There are two entity-tag 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.

A "use" of a validator is either when a client generates a request and includes the validator in a precondition, 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.
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.

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

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.

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.
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 might 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.

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

Consider a resource that is subject to content negotiation (Section 5 of [Part3]), and where the representations returned upon a GET request vary based on the Accept-Encoding request header field (Section 6.3 of [Part3]):

>> 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]) apply only during message transfer and do not require distinct entity-tags.

3. Precondition Header Fields

This section defines the syntax and semantics of HTTP/1.1 header fields for applying preconditions on requests.

3.1. If-Match

The "If-Match" header field MAY be used to make a request method conditional on the current existence or value of an entity-tag for one or more representations of the target resource. If-Match is generally useful for resource update requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource (i.e., the "lost update" problem). An If-Match field-value of "*" places the precondition on the existence of any current representation for the target resource.

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

If any of the entity-tags listed in the If-Match field value match (as per Section 2.2.3) the entity-tag of the selected representation for the target resource, or if "*" is given and any current representation exists for the target resource, then the server MAY perform the request method as if the If-Match header field was not present.
If none of the entity-tags match, or if "*" is given and no current representation exists, the server MUST NOT perform the requested method. Instead, the server MUST respond with the 412 (Precondition Failed) status code.
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 field MUST be ignored.
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.

3.2. If-None-Match

The "If-None-Match" header field MAY be used to make a request method conditional on not matching any of the current entity-tag values for representations of the target resource. If-None-Match is primarily used in conditional GET requests to enable efficient updates of cached information with a minimum amount of transaction overhead. A client that has one or more representations previously obtained from the target resource can send If-None-Match with a list of the associated entity-tags in the hope of receiving a 304 response if at least one of those representations matches the selected representation.
If-None-Match MAY also be used with a value of "*" to prevent an unsafe request method (e.g., PUT) from inadvertently modifying an existing representation of the target resource when the client believes that the resource does not have a current representation. This is a variation on the "lost update" problem that might arise if more than one client attempts to create an initial representation for the target resource.

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

If any of the entity-tags listed in the If-None-Match field-value match (as per Section 2.2.3) the entity-tag of the selected representation, or if "*" is given and any current representation exists for that resource, then the server MUST NOT perform the requested method. Instead, if the request method was GET or HEAD, the server SHOULD respond with a 304 (Not Modified) status code, including the cache-related header fields (particularly ETag) of the selected representation that has a matching entity-tag. 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 field MUST be ignored. (See Section 2.2.4 for a discussion of server behavior when both If-Modified-Since and If-None-Match appear in the same request.)
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.

3.3. If-Modified-Since

The "If-Modified-Since" header field MAY be used to make a request method conditional by modification date: if the selected representation has not been modified since the time specified in this field, then do not perform the request method; instead, respond as detailed below.

```
If-Modified-Since = 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 field and no Range header field requests that the selected representation be transferred only if it has been modified since the date given by the If-Modified-Since header field. 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 selected representation has been modified since the If-Modified-Since date, the response is exactly the same as for a normal GET.

3. If the selected 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 header field modifies the meaning of If-Modified-Since; see Section 5.4 of [Part5] 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 field instead of a date taken from the Last-Modified header field 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.

3.4. If-Unmodified-Since

The "If-Unmodified-Since" header field MAY be used to make a request method conditional by modification date: if the selected representation has been modified since the time specified in this field, then the server MUST NOT perform the requested operation and MUST instead respond with the 412 (Precondition Failed) status code. If the selected representation has not been modified since the time specified in this field, the server SHOULD perform the request method as if the If-Unmodified-Since header field were not present.

```
If-Unmodified-Since = 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 field) would result in anything other than a 2xx or 412 status code, the If-Unmodified-Since header field SHOULD be ignored.

If the specified date is invalid, the header field MUST be 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.

3.5. If-Range

The If-Range header field provides a special conditional request mechanism that is similar to If-Match and If-Unmodified-Since but specific to HTTP range requests. If-Range is defined in Section 5.3 of [Part5].

4. Status Code Definitions

4.1. 304 Not Modified

The 304 status code indicates that a conditional GET request has been received and would have resulted in a 200 (OK) response if it were not for the fact that the condition has evaluated to false. In other words, there is no need for the server to transfer a representation of the target resource because the client's request indicates that it already has a valid representation, as indicated by the 304 response header fields, and is therefore redirecting the client to make use of that stored representation as if it were the payload of a 200 response. 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]) unless its omission is required by Section 9.3.1 of [Part1].
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 the recipient of a 304 response does not have a cached representation corresponding to the entity-tag indicated by the 304 response, then the recipient MUST NOT use the 304 to update its own cache. If this 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, the recipient MUST disregard the 304 response and repeat the request without any preconditions.
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.

4.2. 412 Precondition Failed

The 412 status code indicates that one or more preconditions given in the request header fields evaluated to false when tested on the server. This response code allows the client to place preconditions on the current resource state (its current representations and metadata) and thus prevent the request method from being applied if the target resource is in an unexpected state.

5. IANA Considerations

5.1. Status Code Registration

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 4.1
412	Precondition Failed	Section 4.2

5.2. Header Field Registration

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]):

Header Field Name	Protocol	Status	Reference
ETag	http	standard	Section 2.2
If-Match	http	standard	Section 3.1
If-Modified-Since	http	standard	Section 3.3
If-None-Match	http	standard	Section 3.2
If-Unmodified-Since	http	standard	Section 3.4
Last-Modified	http	standard	Section 2.1

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

6. Security Considerations

No additional security considerations have been identified beyond those applicable to HTTP in general [Part1].

7. Acknowledgments

8. References

8.1. Normative References

[Part1]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", Internet-Draft draft-ietf-httpbis-p1-messaging-14, April 2011.
[Part3]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 3: Message Payload and Content Negotiation", Internet-Draft draft-ietf-httpbis-p3-payload-14, April 2011.
[Part5]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 5: Range Requests and Partial Responses", Internet-Draft draft-ietf-httpbis-p5-range-14, April 2011.
[Part6]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Nottingham, M. and J. F. Reschke, "HTTP/1.1, part 6: Caching", Internet-Draft draft-ietf-httpbis-p6-cache-14, April 2011.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC5234]	Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

8.2. Informative References

[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
[RFC3864]	Klyne, G., Nottingham, M. and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.
[RFC4918]	Dusseault, L.M., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

Appendix A. Changes from RFC 2616

Allow weak entity-tags in all requests except range requests (Sections 2.2.2 and 3.2).
Change ABNF productions for header fields to only define the field value. (Section 3)

Appendix B. Collected ABNF

```
ETag = entity-tag

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

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

Last-Modified = 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)

Appendix C.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

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

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.

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

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.

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

Closed issues:

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

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

* Reference RFC 3984, and update header field registrations for header fields defined in this document.

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

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"

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

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 field value format definitions.

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

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

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

Appendix 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"

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

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"

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

No significant changes.

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

No significant changes.

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

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"

Appendix C.13. Since draft-ietf-httpbis-p4-conditional-11

None.

Appendix C.14. Since draft-ietf-httpbis-p4-conditional-12

Closed issues:

- *http://tools.ietf.org/wg/httpbis/trac/ticket/224: "Header Classification"

Appendix C.15. Since draft-ietf-httpbis-p4-conditional-13

Closed issues:

- *http://tools.ietf.org/wg/httpbis/trac/ticket/89: "If-* and entities"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/101: "Definition of validator weakness"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/276: "untangle ABNFs for header fields"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/269: "ETags and Quotes"

Index

Authors' Addresses

Roy T. Fielding editor Fielding Adobe Systems Incorporated 345 Park Ave San Jose, CA 95110 USA EMail: fielding@gbiv.com URI: http://roy.gbiv.com/

Jim Gettys 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 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 Frystyk Microsoft Corporation 1 Microsoft Way Redmond, WA 98052 USA EMail: henrikn@microsoft.com

Larry Masinter Masinter Adobe Systems Incorporated 345 Park Ave San Jose, CA 95110 USA EMail: LMM@acm.org URI: http://larry.masinter.net/

Paul J. Leach Leach Microsoft Corporation 1 Microsoft Way Redmond, WA 98052 EMail: paulle@microsoft.com

Tim Berners-Lee 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 Lafon 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 Reschke greenbytes GmbH Hafenweg 16 Muenster, NW 48155 Germany Phone: +49 251 2807760 EMail: julian.reschke@greenbytes.de URI: http://greenbytes.de/tech/webdav/