Network Working Group
Internet-Draft
Updates: 2616 (if approved)
Intended status: Informational
Expires: March 26, 2007

J. Reschke
greenbytes

September 22, 2006

The Hypertext Transfer Protocol (HTTP) Entity Tag ("ETag") Response Header in Write Operations draft-reschke-http-etag-on-write-02

Status of this Memo

Copyright Notice

Abstract

The Hypertext Transfer Protocol (HTTP) specifies a state identifier, called "Entity Tag", to be returned in the "ETag" response header. However, the description of this header for write operations such as PUT is incomplete, and has caused confusion among developers and protocol designers, and potentially interoperability problems.

This document explains the problem in detail and suggests both a clarification for a revision to the HTTP/1.1 specification (RFC2616) and a new header for use in responses, making HTTP entity tags more useful for user agents that want to avoid round-trips to the server after modifying a resource.

Editorial Note (To be removed by RFC Editor before publication)

Distribution of this document is unlimited. Please send comments to the Hypertext Transfer Protocol (HTTP) mailing list at ietf-http-wg@w3.org [1], which may be joined by sending a message with subject "subscribe" to ietf-http-wg-request@w3.org [2].

Discussions of the HTTP working group are archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

XML versions, latest edits and the issues list for this document are available from <http://greenbytes.de/tech/webdav/#draft-reschke-http-etag-on-write-latest>.

Table of Contents

1. Introduction

The Hypertext Transfer Protocol (HTTP) specifies a state identifier, called "Entity Tag", to be returned in the "ETag" response header (see [RFC2616], Section 14.19). However, the description of this header for write operations such as PUT is incomplete, and has caused confusion among developers and protocol designers, and potentially interoperability problems.

This document explains the problem in detail and suggests both a clarification for a revision to [RFC2616] and a new header for use in responses, making HTTP entity tags more useful for user agents that want to avoid round-trips to the server after modifying a resource.

1.1. Questions out of Scope

Note that there is a related problem: modifying content-negotiated resources. Here the consensus seems to be simply not to do it. Instead, the origin server should reveal specific URIs of content that is not content-negotiated in the Content-Location response header ([RFC2616], Section 14.14), and user agents should use this more specific URI for authoring. Thus, the remainder of this document will focus on resources for which no content negotiation takes place.

Another related question is the usability of the weak entity tags for authoring (see [RFC2616], Section 13.3.3). Although this document focuses on the usage of strong entity tags, it is believed that the changes suggested in this document could be applied to weak entity tags as well.

1.2. The Interoperability Problem

For a long time, nobody realized that there was a problem at all, or those who realized preferred to ignore it.

Server implementers added code that would return the new value of the "ETag" header in a response to a successful PUT request. After all, the client could be interested in it.

User agent developers in turn were happy to get a new "ETag" value, saving a subsequent HEAD request to retrieve the new entity tag.

However, at some point of time, potentially during a Web Distributed Authoring and Versioning (WebDAV, [RFC2518]) interoperability event, client programmers asked server programmers to always return "ETag" headers upon PUT, never ever to change the entity tag without "good reason", and - by the way - always to guarantee that the server

stores the new content octet-by-octet.

From the perspective of client software that wants to treat an HTTP server as a file system replacement, this makes a lot of sense. After all, when one writes to a file one usually expects the file system to store what was written, and not to unexpectedly change the contents.

However, in general, an HTTP server is not a file system replacement. There may be some that have been designed that way, and some that expose some parts of their namespace that have this quality. But in general, HTTP server implementers have a lot of freedom in how resources are implemented. Indeed, this flexibility is one of the reasons for HTTP's success, allowing it to be used for a wide range of tasks, of which replacing file systems is just one (and not necessarily the most interesting one).

In particular:

- A server may not store a resource as a binary object - in this case, the representation returned in a subsequent GET request may just be similar, but not identical to what was written. Good examples are servers that use HTTP to access XML data ([XCAP]), Calendaring data ([CALDAV]) or newsfeed data ([APP]).

- A server may change the data being written on purpose, while it's being written. Examples that immediately come to mind are keyword substitution in a source control system, or filters that remove potentially insecure parts out of HTML pages.

Furthermore:

- An "unrelated" method such as WebDAV's PROPPATCH (see [RFC2518], Section 8.2) may affect the entity body and therefore the entity tag in an unexpected way, because the server stores some or all of the WebDAV properties inside the entity body (for instance, GPS information inside a JPG image file).

As long as servers store the content octet-by-octet, and return exactly what the client wrote, there is no problem at all.

Things get more interesting when a server does change the content, such as in the "simple authoring" example given in Appendix A.1. Here, the server does change the content upon writing to the resource, yet no harm is done, because the final state of the resource on the server does not depend on the client being aware of that.

All of the content rewriting examples mentioned above have this quality: the client can safely continue to edit the entity it sent, because the result of the transformation done by the server will be the same in the end. Formally, if we call the server-side transformation "t", the initial content "c", and the client-side editing steps "e1" and "e2", then

t(e2(e1(c))) = t(e2(t(e1(c))))

e.g., it's irrelevant whether the client obtained the current entity body before doing its second edit.

[[example.for.non.safe.rewrite: Question: does anybody know a real-world example for server-side content rewriting where the above is not true? --julian.reschke@greenbytes.de]]

Problems will only occur if the client uses the entity body it sent, and the entity tag it obtained in return, in subsequent requests that only transfer part of the entity body, such as GET or PUT requests using the "Range" request header (see [RFC2616], Section 14.35).

Furthermore, some clients need to expose the actual contents to the end user. These clients will have to ensure that they really have the current representation.

Entity bodies (and thus entity tags) changing due to side effects of seemingly unrelated requests are indeed a problem, as demonstrated in Appendix A.2, and this specification proposes a way to resolve this in Section 3.

1.3. Analysis of RFC2616's Definitions

There are several places in the HTTP/1.1 specification ([RFC2616]) mentioning the "ETag" response header.

Let us start with the header definition in Section 14.19:

> The ETag response-header field provides the current value of the entity tag for the requested variant. Sections 14.24, 14.26 and 14.44 describe the headers used with entity tags. The entity tag MAY be used for comparison with other entities from the same resource (see Section 13.3.3).

The meaning of a "response-header" in turn is defined in Section 6.2:

> The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the

> server and about further access to the resource identified by the Request-URI.

The "ETag" response header itself is mentioned mainly in the context of cache validation, such as in Section 13.3.2. What is missing is a coherent description on how the origin server can notify the user-agent when the entity tag changes as result of a write operation, such as PUT.

Indeed, the definition of the 201 Created status code mentions entity tags (Section 10.2.2):

> A 201 response MAY contain an ETag response header field indicating the current value of the entity tag for the requested variant just created, see Section 14.19.

The "ETag" response header is mentioned again in the definition of 206 Partial Content (Section 10.2.7) and 304 Not Modified (Section 10.3.5), but notably missing are statements about other 2xx series status codes that can occur upon a successful PUT operation, such as 200 OK (Section 10.2.1) and 204 No Content (Section 10.2.5).

Summarizing, the specification is a bit vague about what an ETag response header upon a write operation means, but this problem is somewhat mitigated by the precise definition of a response header. A proposal for enhancing [RFC2616] in this regard is made in Section 3 below.

1.4. Prior Work

While working on the revision of [RFC2518], the IETF WebDAV working group realized that this is a generic problem that needs attention independently of WebDAV. An initial attempt was made with [draft-whitehead-http-etag] in February 2006, but no progress was made since.

At the time of this writing in August 2006, two specifications based on HTTP were under IESG review, taking two opposite approaches:

- Section 8.5 of [XCAP] makes it a MUST-level requirement to return an entity tag upon PUT, even though the very nature of an XCAP server will cause it to rewrite contents (due to its XML-based storage).

- Section 5.3.4 of [CALDAV] explicitly forbids ("MUST NOT") returning an entity tag upon PUT if the content was rewritten.

In essence, this makes it impossible to implement an HTTP resource

that conforms to both specifications. Due to the differing use cases of XCAP and CalDAV, this may not be a problem in practice, but the disagreement in itself is scary. Publication of these specifications on the standards track will make it much harder for future protocols to deal with this topic in a meaningful way (comments were sent during IETF Last Call for CalDAV, see <http://www1.ietf.org/mail-archive/web/ietf/current/msg43021.html>).

2. Notational Conventions

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].

The terminology used here follows and extends that in the HTTP specification [RFC2616], notably the augmented Backus-Naur Form (BNF) defined in Section 2.1 of that document.

3. Clarifications on the Behavior of the 'ETag' Response Header

This section describes a minimal change to [RFC2616], proposed in <http://lists.w3.org/Archives/Public/ietf-http-wg/2006JanMar/0003.html>.

At the end of Section 10.2 of [RFC2616], add:

> The response MAY contain an ETag response header field indicating the current value of the entity tag (Section 14.19) for the requested variant. The value SHOULD be the same as the one returned upon a subsequent HEAD request addressing the same variant.

In Section 10.2.1 of [RFC2616], remove:

> A 201 response MAY contain an ETag response header field indicating the current value of the entity tag for the requested variant just created, see Section 14.19.

In essence, this moves the statement about entity tags in write operations from the specific case of status 201 Created into the more generic description of the 2xx series status codes.

[[rcf2616.enhancements: Should "requested variant" be clarified in the context of write operations? --julian.reschke@greenbytes.de]]

4. The 'Entity-Transform' Header

The 'Entity-Transform' entity header provides information about whether a transformation has been applied to an entity body.

When used in an HTTP request, its meaning is undefined. In an HTTP response, it provides information whether the server has applied a transformation when the entity was stored last.

In general, entity headers may be stored in intermediates. The main use of this header however applies to the HTTP PUT method, of which by default the results are not cacheable (see [RFC2616], Section 9.6). In addition, the value format is defined so that a client can reliably detect whether the information is fresh.

Format:

```
Entity-Transform = "Entity-Transform" ":" entity-transform-spec
entity-transform-spec = entity-transform-keyword SP entity-tag
entity-transform-keyword = "identity" | "unspecified"
; entity-tag: defined in [RFC2616], Section 3.11
```

The entity-tag specifies the entity body to which this information applies.

An entity-transform-keyword of "identity" specifies that the origin server has stored the entity octet-by-octet, thus the user agent MAY use a local copy of the entity body with the given entity-tag for subsequent requests that rely on octet-by-octet identity (such as a PUT with "Range" request header).

Both the absence of this response header and any entity-transform-keyword value other than "identity" specify that the origin server may have transformed the entity before storage, thus a subsequent retrieval will not necessarily return an exact copy of the submitted entity.

4.1. Examples

Content was stored octet-by-octet:

```
HTTP/1.1 200 OK
ETag: "1"
Entity-Transform: identity "1"
```

Content was transformed:

```
HTTP/1.1 200 OK
ETag: "2"
Entity-Transform: unspecified "2"
```

Response containing a stale "Entity-Transform" header:

```
HTTP/1.1 200 OK
ETag: "3"
Entity-Transform: unspecified "2"
```

Note that in this case the newly assigned entity tag and the entity tag returned in "Entity-Transform" do not match, thus the client is aware that the header value is stale and can't be used.

5. How This Helps

The clarification of [RFC2616] (see Section 3) makes it clear that user agents can use "ETag" headers obtained in write operations, as long as they do not require octet-by-octet identity. In particular, a new entity tag can be returned for any method, such as a WebDAV PROPPATCH (see [RFC2518], Section 8.2. This helps dealing with the problem described in Appendix A.2. See Appendix A.3 for details.

The addition of the "Entity-Transform" header (see Section 4) enables origin servers to signal that they stored an exact copy of the content, thus allowing clients not to refetch the content. Note that by default (in absence of the response header), a client can not make any assumptions about the server's behavior in this regard. Thus clients will only benefit from new servers explicitly setting the new header.

6. IANA Considerations

This document specifies the new HTTP header listed below.

Header field name: Entity-Transform

Applicable protocol: http

Status: informational

Author/Change controller: IETF

Specification document: Section 4 of this specification

7. Security Considerations

This specification introduces no new security considerations beyond those discussed in Section 15 of [RFC2616].

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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.

8.2. Informative References

[APP] Gregorio, J., Ed. and B. de hOra, Ed., "The Atom Publishing Protocol", draft-ietf-atompub-protocol-10 (work in progress), September 2006.

[CALDAV] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", draft-dusseault-caldav-15 (work in progress), September 2006.

[HTML] Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01 Specification", W3C REC-html401-19991224, December 1999, <http://www.w3.org/TR/html401/>.

[RFC2518] Goland, Y., Whitehead, E., Faizi, A., Carter, S., and D. Jensen, "HTTP Extensions for Distributed Authoring -- WEBDAV", RFC 2518, February 1999.

[XCAP] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-ietf-simple-xcap-11 (work in progress), May 2006.

[draft-whitehead-http-etag] Whitehead, J., "Design Considerations for State Identifiers in HTTP and WebDAV",

draft-whitehead-http-etag-00 (work in progress),
February 2006.

URIs

[1] <mailto:ietf-http-wg@w3.org>

[2] <mailto:ietf-http-wg-request@w3.org?subject=subscribe>

Appendix A. Use Cases

A.1. Simple Authoring

Let us consider a server not having the quality of preserving octet-by-octet identity, for instance because of SVN-style keyword expansion in text content (<http://svnbook.red-bean.com/en/1.2/svn-book.html#svn.advanced.props.special.keywords>).

In this case, the client has previously retrieved the representation for <http://example.com/test>, and the server has returned the ETag "1":

```
>> Request (1)

GET /test HTTP/1.1
Host: example.com

>> Response (1)

HTTP/1.1 200 OK
Content-Type: text/plain
ETag: "1"

# $Revision: 1 $
Sample text.
```

The client now wants to update the resource. To avoid overwriting somebody else's changes, it submits the PUT request with the HTTP "If-Match" request header (see [RFC2616], Section 14.24):

>> Request (2)

```
PUT /test HTTP/1.1
Host: example.com
If-Match: "1"

# $Revision: 1 $
New sample text.
```

If the resource was modified in the meantime, the server will reject the request with a 412 Precondition Failed status:

>> Response (2a)

```
HTTP/1.1 412 Precondition Failed
Content-Type: text/plain

Precondition Failed: entity tag supplied in If-Match request header
did not match current.
```

In this case, the client usually has take care of merging the changes made locally with those made on the server ("Merge Conflict").

If there was no overlapping update, the server will execute the request and return a new entity tag:

>> Response (2b)

```
HTTP/1.1 200 OK
Content-Type: text/plain
ETag: "2"
```

Note however, that at this point the client knows the new entity tag, but doesn not know the current representation, which will have been updated by the server to:

```
# $Revision: 2 $
New sample text.
```

What seems to be a problem at first may not be a real problem in practice. Let us assume that the client continues editing the resource, using the entity tag obtained from the previous request, but editing the entity it last sent:

```
>> Request (3)

PUT /test HTTP/1.1
Host: example.com
If-Match: "2"

# $Revision: 1 $
A third attempt.
```

Assuming there was no overlapping update, the PUT request will succeed:

```
>> Response (3)

HTTP/1.1 200 OK
Content-Type: text/plain
ETag: "3"
```

Note that the only problem here is that the client doesn't have an exact copy of the entity it's editing. However, from the server's point of view this is entirely irrelevant, because the "Revision" keyword will be automatically updated upon every write anyway.

In any case, the final contents will be:

```
# $Revision: 3 $
A third attempt.
```

A.2. Setting both Content and Metadata

In this example, the server exposes data extracted from the HTML <title> element ([HTML], Section 7.4.2) as a custom WebDAV property ([RFC2518], Section 4), allowing both read and write access.

In the first step, the client obtains the current representation for <http://example.com/test.html>:

```
>> Request (1)

GET /test.html HTTP/1.1
Host: example.com
```

```
>> Response (1)

HTTP/1.1 200 OK
Content-Type: text/html
ETag: "A"

<html>
  <head>
  </head>
  <body>
  </body>
</html>
```

Next, it adds one paragraph to the <body> element, and gets back a new entity tag:

```
>> Request (2)

PUT /test.html HTTP/1.1
Host: example.com
If-Match: "A"

<html>
  <head>
  </head>
  <body>
    <p>First paragraph.</p>
  </body>
</html>

>> Response (2)

HTTP/1.1 200 OK
ETag: "B"
```

Next, the client sets a custom "title" property (see [RFC2518], Section 8.2):

```
>> Request (3)

PROPPATCH /test.html HTTP/1.1
Host: example.com
Content-Type: application/xml

<proppatch xmlns="DAV:">
  <set>
    <prop>
      <title xmlns="http://ns.example.org/">
        A sample document
      </title>
    </prop>
  </set>
</proppatch>

>> Response (3)

HTTP/1.1 207 Multi-Status
Content-Type: application/xml

<multistatus xmlns="DAV:">
  <response>
    <href>/test.html</href>
    <propstat>
      <prop>
        <title xmlns="http://ns.example.org/"/>
      </prop>
      <status>HTTP/1.1 2OO OK</status>
    </propstat>
  </response>
</multistatus>
```

The server knows how to propagate property changes into the HTML content, so it updates the entity by adding an HTML title document accordingly. This causes the entity tag changing to "C".

The new entity body is shown below, but the client does not realize that it did change at all.

```
<html>
  <head>
    <title>A sample document</title>
  </head>
  <body>
    <p>First paragraph.</p>
  </body>
</html>
```

A subsequent attempt by the client to update the entity body will fail, unless it realizes that changing WebDAV properties may affect the entity as well. In this case, it would have had to get the current entity tag before proceeding. Of course, this introduces an additional round-trip, and a timing window during which overlapping updates by other clients would go unnoticed.

A.3. Setting both Content and Metadata, new style

Below we repeat the example from above (Appendix A.2), but here the origin server returns entity tags for all write operations, and the user agent knows how to deal with them. That is, both take advantage of [RFC2616] already allows.

```
>> Request (3)

PROPPATCH /test.html HTTP/1.1
Host: example.com
Content-Type: application/xml

<proppatch xmlns="DAV:">
  <set>
    <prop>
      <title xmlns="http://ns.example.org/">
        A sample document
      </title>
    </prop>
  </set>
</proppatch>

>> Response (3)

HTTP/1.1 207 Multi-Status
Content-Type: application/xml
ETag: "C"

<multistatus xmlns="DAV:">
  <response>
    <href>/test.html</href>
    <propstat>
      <prop>
        <title xmlns="http://ns.example.org/"/>
      </prop>
      <status>HTTP/1.1 2OO OK</status>
    </propstat>
  </response>
</multistatus>
```

As before, this causes the entity to change, and a new entity tag to be assigned. But in this case, the origin server actually notifies the client of the changed state by including the "ETag" response header.

The client now will be aware that the requested entity change, and can use the new entity tag in subsequent requests (potentially after refreshing the local copy).

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

B.1. Since draft-reschke-http-etag-on-write-00

Add and resolves issues "entity-header" and "extensibility", by removing the extension hooks, and by redefining the header to it can be used as an Entity header.

B.2. Since draft-reschke-http-etag-on-write-01

Update APP and CALDAV references. Remove RFC3986 reference (not needed anymore after the simplication in draft 01). Fix typo in header description ("submitted entity", not "stored entity"). Remove comparison about how XCAP and CALDAV profile RFC2616: after all, both mandate a behaviour that was legal but optional before. Add "Updates: RFC2616".

Appendix C. Open issues (to be removed by RFC Editor prior to publication)

C.1. edit

Type: edit

julian.reschke@greenbytes.de (2006-08-10): Umbrella issue for editorial fixes/enhancements.

Author's Address

Julian F. Reschke
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/

Full Copyright Statement

Intellectual Property

Acknowledgment

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