XEP-0150: Use of Entity Tags in XMPP Extensions

Joe Hildebrand
mailto:jhildebr@cisco.com
xmpp:hildjj@jabber.org

Peter Saint-Andre
mailto:xsf@stpeter.im
xmpp:peter@jabber.org
http://stpeter.im/

2005-08-09
Version 0.2

Status	Type	Short Name
Deferred	Informational	N/A

This document defines best practices for the use of Entity Tags in XMPP extensions.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF’s Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1 Introduction

When an XMPP instant messaging client connects to its server, it typically retrieves a great deal of information, such as a user's roster, privacy lists, and service discovery information. However, this information may not have changed since the client last retrieved the data. In order to improve client start-up time or conserve bandwidth in certain environments, it would be desirable if the client could cache the information and retrieve it only if it has changed.
Such a mechanism exists as part of RFC 2616 [1] in the form of the Entity Tags, which are included in the ETag and If-None-Match headers used for HTTP caching. Since Stanza Headers and Internet Metadata (XEP-0131) [2] enables an XMPP entity to communicate any HTTP header, it should be possible to re-use existing Entity Tag semantics for caching information sent over an XMPP network. This document defines best practices for such functionality, which could be used between any two XMPP entities that support SHIM headers (though it is envisioned to be most useful for clients that retrieve information from servers and services).

2 Requirements

This document addresses the following requirements:

1. Enable caching of information sent over an XMPP network (i.e., retrieval of information only if it has changed since it was last retrieved).
2. Re-use existing HTTP Entity Tag semantics.

3 Concepts and Approach

In HTTP, an "entity" is the information transferred as the payload of a request or response, and an "Entity Tag" is an opaque string that uniquely identifies that payload. For example, when an HTTP server sends an entity in an HTTP response, it can include an ETag header that identifies the payload as cacheable, and the client can cache that entity; in future requests, the client can include the same value in an If-None-Match header and the server will either return an HTTP 304 ("Not Modified") error if the entity has not changed or return the entity in the HTTP response if it has changed. (Note: For information about the semantics of Entity Tags, the ETag header, and the If-None-Match header, refer to Sections 3.11, 14.19, and 14.26 respectively of RFC 2616.)
The basic concept behind XMPP Entity Tag use is semantically equivalent to the use in HTTP (although we use the term "data object" to refer to the payload); this is made possible by a straightforward application of SHIM headers as specified in XEP-0131. In the context of an IQ

[1] RFC 2616: Hypertext Transport Protocol -- HTTP/1.1 <http://tools.ietf.org/html/rfc2616>.
[2] XEP-0131: Stanza Headers and Internet Metadata <https://xmpp.org/extensions/xep-0131.html>.

request-response interaction, the responding entity will include an ETag SHIM header in its IQ response (indicating that the data object can be cached), the requesting entity will include that identical value in an If-None-Match SHIM header when it queries the server for the same data object, and the responding entity will return either an IQ result with the changed data object or an IQ error indicating that the data object has not changed.
Here is an example of such a request-response flow (the example is that of roster retrieval):

Listing 1: Client Requests Roster

```
<iq type='get'
    from='juliet@capulet.com/balcony'
    id='roster1'>
 <query xmlns='jabber:iq:roster'/>
</iq>
```

Listing 2: Server Returns Roster Including ETag Header

```
<iq type='result'
    to='juliet@capulet.com/balcony'
    id='roster1'>
 <query xmlns='jabber:iq:roster'>
    <item jid='romeo@montague.net' name='Romeo'>
      <group>Friends</group>
    </item>
    <item jid='nurse@capulet.com' name='Nurse'>
      <group>Servants</group>
    </item>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='ETag'>some-long-opaque-string</header>
    </headers>
 </query>
</iq>
```

Listing 3: Client Requests Roster Including If-None-Match Header

```
<iq type='get'
    from='juliet@capulet.com/balcony'
    id='roster2'>
 <query xmlns='jabber:iq:roster'>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='If-None-Match'>some-long-opaque-string</header>
    </headers>
 </query>
</iq>
```

If the responding entity does not understand the If-None-Match header or does not handle Entity Tags for the namespace in the request, it MUST ignore the header and return whatever information it would have returned if the header had not been present.

If the responding entity determines that the requested information has not changed since it was last retrieved by the requesting entity, then it MUST return a <not-modified/> error corresponding to the HTTP 304 error returned by HTTP entities that support the ETag header:

Listing 4: Responding Entity Returns Not Modified Error

```
<iq type='error'
    to='juliet@capulet.com/balcony'
    id='roster2'>
 <query xmlns='jabber:iq:roster'>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='If-None-Match'>some-long-opaque-string</header>
    </headers>
 </query>
  <error code='304' type='modify'>
    <not-modified xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Note: The <not-modified/> error condition is not specified as a stanza error condition in RFC 3920 [3] and an error code of 304 was not included in the older Jabber error codes (see Error Condition Mappings (XEP-0086) [4]). However, the <not-modified/> error condition is included in XMPP Core [5].
Note: In HTTP, an Entity Tag may be either "strong" or "weak" (see Section 13.3.3 of RFC 2616); Entity Tags as used in XMPP extensions MUST be considered strong rather than weak.
Note: The ETag and If-None-Match headers SHOULD be used only in <iq/> stanzas, although they MAY be used in <message/> stanza interactions if IQ request-response semantics are not appropriate, for example in SOAP over XMPP (XEP-0072) [6] and in certain applications that use Data Forms (XEP-0004) [7].

4 Use Cases

4.1 Caching and Retrieving Rosters

As specified in XMPP IM [8], an XMPP instant messaging client will typically store its "roster" (contact list) on the server so that any connecting client for that account can retrieve the roster at will. Since RFC 6121 defines no upper limit on the number of items allowed in the roster, it is possible for a roster to become quite large (e.g., there are known cases of rosters

[3] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.
[4] XEP-0086: Error Condition Mappings <https://xmpp.org/extensions/xep-0086.html>.
[5] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[6] XEP-0072: SOAP over XMPP <https://xmpp.org/extensions/xep-0072.html>.
[7] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.
[8] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

with more than 1,000 items). Therefore a server may support Entity Tag functionality with regard to roster management. The process is as follows.
First, the client requests its roster:

Listing 5: Client Requests Roster

```
<iq type='get'
    from='juliet@capulet.com/balcony'
    id='roster1'>
 <query xmlns='jabber:iq:roster'/>
</iq>
```

If the server supports Entity Tag functionality for rosters, it SHOULD include an ETag SHIM header in its response (although it MAY decide not to include the header if the roster is deemed to be not worth caching because it is so small):

Listing 6: Server Returns Roster Including ETag Header

```
<iq type='result'
    to='juliet@capulet.com/balcony'
    id='roster1'>
 <query xmlns='jabber:iq:roster'>
    <item jid='romeo@montague.net' name='Romeo'>
      <group>Friends</group>
    </item>
    <item jid='nurse@capulet.com' name='Nurse'>
      <group>Servants</group>
    </item>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='ETag'>some-long-opaque-string</header>
    </headers>
 </query>
</iq>
```

The client would then cache that roster and associate the included Entity Tag with that cached copy. In order to subsequently retrieve the roster, the client would include the last known Entity Tag value with the request in an If-None-Match SHIM header:

Listing 7: Client Requests Roster, Including If-None-Match Header

```
<iq type='get'
    from='juliet@capulet.com/balcony'
    id='roster2'>
 <query xmlns='jabber:iq:roster'>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='If-None-Match'>some-long-opaque-string</header>
    </headers>
 </query>
```

```
</iq>
```

If the roster did not change since the client last retrieved the roster and the server supports Entity Tags for the 'jabber:iq:roster' namespace, the server MUST return a <not-modified/> error:

Listing 8: Server Returns Not Modified Error

```
<iq type='error'
    to='juliet@capulet.com/balcony'
    id='roster2'>
 <query xmlns='jabber:iq:roster'>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='ETag'>some-long-opaque-string</header>
    </headers>
 </query>
  <error code='304' type='modify'>
    <not-modified xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the roster has changed, the provided Entity Tag value is not valid, or the server does not support Entity Tags, it MUST return the roster as if the If-None-Match header was not included:

Listing 9: Server Returns Roster

```
<iq type='result'
    to='juliet@capulet.com/balcony'
    id='roster2'>
 <query xmlns='jabber:iq:roster'>
    <item jid='romeo@montague.net' name='Romeo'>
      <group>Friends</group>
    </item>
    <item jid='nurse@capulet.com' name='Nurse'>
      <group>Servants</group>
    </item>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='ETag'>some-new-opaque-string</header>
    </headers>
 </query>
</iq>
```

4.2 Caching and Retrieving Privacy Lists

The payloads exchanged in the Privacy Lists (XEP-0016) [9] protocol can also be quite large. Therefore a server might want to support Entity Tags in the context of privacy list management. The process is as follows.

[9] XEP-0016: Privacy Lists <https://xmpp.org/extensions/xep-0016.html>.

First, a client requests its privacy lists:

Listing 10: Client Requests Privacy List

```
<iq type='get'
    from='romeo@montague.net/orchard'
    id='getlist1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='special'/>
  </query>
</iq>
```

The server returns the list and includes an Entity Tag in an ETag SHIM header.

Listing 11: Server Returns Privacy List Including ETag Header

```
<iq type='result'
    to='romeo@example.net/orchard'
    id='getlist1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='special'>
      <item type='jid'
            value='juliet@example.com'
            action='allow'
            order='6'/>
      <item type='jid'
            value='benvolio@example.org'
            action='allow'
            order='7'/>
      <item type='jid'
            value='mercutio@example.org'
            action='allow'
            order='42'/>
      <item action='deny' order='666'/>
    </list>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='ETag'>some-long-opaque-string</header>
    </headers>
  </query>
</iq>
```

Later, the client requests the same privacy list again and includes the provided Entity Tag in an If-None-Match SHIM header:

Listing 12: Client Requests Privacy List Including If-None-Match Header

```
<iq type='get'
    from='romeo@montague.net/orchard'
    id='getlist2'>
```

```
  <query xmlns='jabber:iq:privacy'>
    <list name='special'/>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='If-None-Match'>some-long-opaque-string</header>
    </headers>
  </query>
</iq>
```

If the privacy list did not change since the client last retrieved it and the server supports Entity Tags for the 'jabber:iq:privacy' namespace, the server MUST return a <not-modified/> error:

Listing 13: Server Returns Not Modified Error

```
<iq type='error'
    to='romeo@example.net/orchard'
    id='getlist2'>
 <query xmlns='jabber:iq:privacy'>
    <list name='special'/>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='ETag'>some-long-opaque-string</header>
    </headers>
 </query>
  <error code='304' type='modify'>
    <not-modified xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the privacy list has changed, the provided Entity Tag value is not valid, or the server does not support Entity Tags, it MUST return the privacy list as if the If-None-Match header was not included:

Listing 14: Server Returns Privacy List

```
<iq type='result'
    to='romeo@example.net/orchard'
    id='getlist2'>
  <query xmlns='jabber:iq:privacy'>
    <list name='special'>
      <item type='jid'
            value='juliet@example.com'
            action='allow'
            order='6'/>
      <item type='jid'
            value='benvolio@example.org'
            action='allow'
            order='7'/>
      <item type='jid'
            value='mercutio@example.org'
```

```
            action='allow'
            order='42'/>
      <item action='deny' order='666'/>
    </list>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='ETag'>some-new-opaque-string</header>
    </headers>
  </query>
</iq>
```

4.3 Discovering Support

XEP-0131 specifies how support for a particular SHIM header can be explicitly determined using Service Discovery (XEP-0030) [10]. To review, the protocol flow between a client and a server is as follows:

Listing 15: Client Queries Server Regarding SHIM Header Support

```
<iq from='juliet@capulet.com/balcony'
    to='capulet.com'
    id='header1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/shim'/>
</iq>
```

Listing 16: Server Communicates Supported SHIM Headers

```
<iq from='capulet.com'
    to='juliet@capulet.com/balcony'
    id='header1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/shim'>
    ...
    <feature var='http://jabber.org/protocol/shim#ETag'/>
    <feature var='http://jabber.org/protocol/shim#If-None-Match'/>
    ...
  </query>
</iq>
```

The client now knows that the server supports the ETag and If-None-Match SHIM headers and can proceed accordingly.
Note: If an XMPP entity supports Entity Tags as specified herein, it MUST at a minimum support both the ETag and If-None-Match SHIM headers.
Note: Even if an entity supports the ETag and If-None-Match SHIM headers, it is not required to support Entity Tag functionality for all namespaces. For example, a server could support Entity Tags only for rosters and privacy lists but not for the 'jabber:iq:last' or 'jabber:iq:version'

[10] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

namespaces. Similarly, a Multi-User Chat (XEP-0045) [11] service could support Entity Tags only for room lists (retrieved via a ”disco#items” request) but not for other requests. As noted, if an entity does not support Entity Tags for a given namespace or request, it SHOULD proceed as if the ETag or If-None-Match SHIM header had not been included in the request.
Optionally, an entity MAY communicate the namespaces for which it supports Entity Tag functionality by listing those namespaces in its response to a ”disco#info” request sent to a node of ”http://jabber.org/protocol/shim#ETag”:

Listing 17: Client Queries Server Regarding Supported Entity Tag Namespaces

```
<iq from='juliet@capulet.com/balcony'
    to='capulet.com'
    id='header1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/shim#ETag'/>
</iq>
```

Listing 18: Server Communicates Supported Entity Tag Namespaces

```
<iq from='capulet.com'
    to='juliet@capulet.com/balcony'
    id='header1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/shim#ETag'>
    ...
    <feature var='jabber:iq:roster'/>
    <feature var='jabber:iq:privacy'/>
    ...
  </query>
</iq>
```

Alternatively, as shown above and as used in HTTP, the requesting entity MAY implicitly discover that Entity Tag functionality is supported with regard to a given response entity type if the responding entity includes an ETag SHIM header in its response.

5 Security Considerations

If a malicious entity gains access to a user’s credentials or is able to masquerade as another entity on the network (e.g., as a man in the middle), it could force retrieval of information before it has changed. However, such access would compromise communications in a more serious fashion and corruption of the Entity Tags functionality is insignificant in comparison.

[11] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

6 IANA Considerations

This proposal requires no interaction with the Internet Assigned Numbers Authority (IANA) [12].

7 XMPP Registrar Considerations

This proposal requires no interaction with the XMPP Registrar [13].

8 XML Schema

This document describes best practices for use of XEP-0131 and therefore does not require a dedicated schema.

9 Open Issues

Is the ETag tied to a bare JID or full JID?

[12] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

[13] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.