XEP-0303: Commenting

Justin Karneges
mailto:justin@karneges.com
xmpp:justin@andbit.net

2011-07-28
Version 0.1

Status	Type	Short Name
Deferred	Standards Track	NOT_YET_ASSIGNED

This specification defines a method for commenting.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2024 by the XMPP Standards Foundation (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

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

Commenting is a popular activity on the Internet. Users leave comments on just about anything: blog posts, news articles, product reviews, photos, status updates, etc. Existing commenting solutions often involve proprietary access methods and authentication, and are silo'd off from other services. This specification proposes an open and federated way of commenting. A conversation exists as a set of Publish-Subscribe (XEP-0060) [1] nodes, containing comment items or other activity, and any user with a JID may leave a comment there (per conversation access rules). The protocol is designed to be modern, social, and extensible. Additionally, while the protocol is described in XMPP terms, the core concept is meant to translate easily to the HTTP-based Social Web.

2 Requirements

The following features are required:

- A client MUST be able to efficiently request the most recent portion of a conversation (as well as "page" to further portions) without having to fetch every comment.

- A client MUST be able to efficiently receive updates to a conversation as they happen.

- It MUST be possible to nest comments (comments as replies to other comments) for rendering the conversation as a tree.

- A user MUST be able to mention another user in a comment, and have that user be notified about it.

- A user MUST be able to efficiently track its own history in a conversation using a third-party service. For example, such a service might maintain a web page of the user's activity, or send an SMS message to the user if a reply is discovered.

- To enable future expansion, consideration must be made for allowing non-comment activity in the conversation, even if this specification does not define those activity types.

3 Result Set Management

Publish-Subscribe (XEP-0060) [2] contains an example of how to retrieve items with Result Set Management (XEP-0059) [3], but the specification is light on details. This section gives a more complete description of how RSM is used in conjunction with PubSub.
Clients MAY provide an RSM section in the iq request:

---

[1]XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[2]XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[3]XEP-0059: Result Set Management <https://xmpp.org/extensions/xep-0059.html>.

Listing 1: Requesting items with a maximum of 20 in the result set

```
<iq type="get" from="alice@example.com/1" to="comments.example.com" id
    ="1">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <items node="activity"/>
    <set xmlns="http://jabber.org/protocol/rsm">
      <max>20</max>
    </set>
  </pubsub>
</iq>
```

The server MAY provide an RSM section in the response. This is already documented in XEP-0060. The server can do this regardless of whether or not the client has made the request with RSM.

Use of RSM implies that there is a natural ordering of the items at a node. The criteria used for ordering is to be determined by the node.

RSM and max_items do not mix. If the client provides both, then the server MUST prefer RSM. If the server does not support RSM, then it may honor max_items and return items ordered by newest first (which may not necessarily be the same as the ordering used by RSM).

4 How It Works

4.1 Conversation Nodes

A conversation exists across a set of PubSub nodes, some of which are dynamic:

Node	Natural Sort Order	Description
"info"	N/A	Singleton node containing information about the conversation.
"comments" (dynamic)	modified-ascending	Contains comment items only. This node is primarily used for presenting conversations to the user. It is essentially a subset of the activity node.

Node	Natural Sort Order	Description
"activity" (dynamic)	modified-ascending	Contains all activity items in the conversation, including comments. This node is primarily used for submitting comments and receiving mention events. Item persistence is OPTIONAL. Advanced implementations may choose to maintain full activity history of a conversation and expose it in this node.

The comments and activity nodes share item data. Comments are added to the conversation by publishing to the activity node, yet the comment will also appear in the comments node as a result. In fact, since the activity node is not required to offer item persistence, it is possible that the comment might *only* be retrievable through the comments node. Implementations of this protocol will therefore require tight association between the comments and activity nodes. It is not possible to implement this protocol using a "generic" PubSub service.

A dynamic node accepts additional parameters by appending the parameters to the node name using a "query"-like notation. Parameters and values in the query string MUST be percent-encoded.

Name	Allowed Values	Applies To	Example
"order"	"-created" (sort items by created time, descending)	comments	Node name "comments?order=-created" would present comments in created-descending order.
"parent_ids"	Comma-separated list of parent comment IDs to filter by. An empty value means to include top-level comments.	comments	Node name "comments?order=-created&parent_-ids=1%2C5a%2Co19g%2C" (note last value is empty) would present items in created-descending order, filtered to only include comments that have parent ID "1", "5a", "o19g", or are top-level.

The "activity" node is defined as dynamic to allow for future expansion, even though no dynamic node parameters are defined in this document that apply to it.

Before utilizing additional nodes or parameters not defined in this document, the client SHOULD first determine support via Service Discovery (XEP-0030) [4] or other discovery mechanism. If the server does not support or understand a parameter or value, it SHOULD reject the request. Attempting to service a request by ignoring unsupported parameters will most likely result in incorrect or undesired behavior.

A conversation is accessible through a JID and optionally a node prefix. The prefix is prepended to the desired node name, separated by a '/' character. For example, if the "Coffee Talk" conversation is said to be accessible at the JID "coffeetalk@comments.example.com" without a node prefix, then PubSub interactions are made using the node names defined above (i.e. "info", "comments", etc). If that conversation is instead said to be accessible at the JID "comments.example.com" with node prefix "coffeetalk", then PubSub interactions are made using node names like "coffeetalk/info", "coffeetalk/comments", etc. This allows conversations to be provisioned as either one per JID or many per JID.

4.2 Retrieving Information About the Conversation

Information about a conversation can be obtained by requesting the item from the info node.

Listing 2: Retrieve conversation info

```xml
<iq type="get" from="alice@example.com/1" to="comments.example.com" id="2">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <items node="coffeetalk/info"/>
  </pubsub>
</iq>
```

Listing 3: Server responds

```xml
<iq type="result" from="comments.example.com" to="alice@example.com/1" id="2">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <items node="coffeetalk/info">
      <item id="current">
        <entry xmlns="http://www.w3.org/2005/Atom">
          <title>Coffee Talk</title>
          <summary>A great place to talk about your day.</summary>
          <id>tag:comments.example.com,2011:coffeetalk</id>
          <published>2011-07-01T10:15:00Z</published>
          <updated>2011-07-01T10:15:00Z</updated>
```

---

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

```
            </entry>
         </item>
      </items>
   </pubsub>
</iq>
```

It is also possible to subscribe to the info node to track changes to the conversation information.

4.3 Retrieving Comments in a Conversation

The process for retrieving the first "page" of a conversation and listening for further updates can be summarized as follows:

1. Subscribe to the "comments" node.

2. Retrieve items from the "comments?order=-created" node. Multiple retrieval requests may be required depending on the client display needs (parent_ids can be used to reduce trips, see the Implementation Notes)

3. Updates are pushed via the subscription.

First, the client subscribes to the comments node. A temporary subscription is recommended, so that it is removed if the client goes offline.

Listing 4: Subscribe to comments node

```
<iq type="set" from="alice@example.com/1" to="comments.example.com" id
   ="1">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <subscribe node="coffeetalk/comments" jid="alice@example.com/1"/>
    <options>
      <x xmlns="jabber:x:data" type="submit">
        <field var="FORM_TYPE" type="hidden">
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
             value>
        </field>
        <field var="pubsub#expire"><value>presence</value></field>
      </x>
    </options>
  </pubsub>
</iq>
```

Next, the client retrieves past comments. For simplicity, the example below will fetch the 50 most recently created comments. This would be useful for displaying the comments in a flat list.

Listing 5: Retrieve the most recent comments

```
<iq type="get" from="alice@example.com/1" to="comments.example.com" id
    ="2">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <items node="coffeetalk/comments?order=-created"/>
    <set xmlns="http://jabber.org/protocol/rsm">
      <max>50</max>
    </set>
  </pubsub>
</iq>
```

Listing 6: Server responds

```
<iq type="result" from="comments.example.com" to="alice@example.com/1"
    id="2">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <items node="coffeetalk/comments?order=-created">
      <item id="39267824-cdc8-11df-b1a7-0024bed71c0a">
        <entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="
            http://activitystrea.ms/spec/1.0/">
          <id>1</id>
          <title>Bob posted a comment in the Coffee Talk conversation.
              </title>
          <summary>Bob posted a comment in the Coffee Talk
              conversation.</summary>
          <published>2011-07-01T12:00:00Z</published>
          <updated>2011-07-01T12:00:00Z</updated>
          <author>
            <name>Bob</name>
            <uri>acct:bob@example.com</uri>
            <activity:object-type>person</activity:object-type>
          </author>
          <activity:object>
            <id>1</id>
            <title>This is a nice comment.</title>
            <content type="text/html">This is a nice comment.</summary
                >
            <activity:object-type>comment</activity:object-type>
          </activity:object>
        </entry>
      </item>
      ...
    </items>
    <set xmlns="http://jabber.org/protocol/rsm">
      <first index="0">39267824-cdc8-11df-b1a7-0024bed71c0a</first>
      <last>ac277776-cdd5-11df-92c4-0024bed71c0a</last>
      <count>5</count>
    </set>
  </pubsub>
```

```
</iq>
```

Comments are stored as Activity Streams items in Atom format.

4.4 Submitting a Comment

Comments are submitted by publishing the comment to the activity node.

Listing 7: Publishing a comment

```
<iq type="set" from="alice@example.com/1" to="comments.example.com" id
    ="3">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <publish node="activity">
      <item>
        <entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="
            http://activitystrea.ms/spec/1.0/">
          <id>2</id>
          <title>Alice posted a comment in the Coffee Talk
              conversation.</title>
          <summary>Alice posted a comment in the Coffee Talk
              conversation.</summary>
          <published>2011-07-01T13:00:00Z</published>
          <updated>2011-07-01T13:00:00Z</updated>
          <activity:object>
            <id>2</id>
            <title>This is another nice comment.</title>
            <content type="text/html">This is another nice comment.</
                summary>
            <activity:object-type>comment</activity:object-type>
          </activity:object>
        </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```

Upon receiving the request, the server MUST sanitize the item as necessary before accepting it. In particular, the author information MUST be confirmed or replaced with the information of the user that submitted the comment. If the item is not formatted as a valid Activity Streams Comment, then the request MUST be rejected.

Listing 8: Rejecting invalid comment

```
<iq type="error" from="comments.example.com" to="alice@example.com/1"
    id="3">
  <error type="modify">
```

```
        <bad-request xmlns="urn:ietf:params:xml:ns:xmpp-stanzas"/>
    </error>
</iq>
```

Next, the server SHOULD ensure it has the submitter's user information. This is so when the comment is served to other clients, name and avatar information can be provided as well. This is done by requesting the VCard of the submitter's bare JID using vcard-temp (XEP-0054) [5]. If the server skips this step, the author name SHOULD be the bare JID of the submitter.

Listing 9: Server responds with success

```
<iq type="result" from="comments.example.com" to="alice@example.com/1"
    id="3">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <publish node="activity">
      <item id="0f72afbe-a9d4-11e0-b0bc-0024bed71c0a"/>
    </publish>
  </pubsub>
</iq>
```

Listing 10: Server relays comment to subscribers

```
<message type="headline" from="comments.example.com" to="alice@example
    .com/1">
  <event xmlns="http://jabber.org/protocol/pubsub#event">
    <items node="comments">
      <item id="0f72afbe-a9d4-11e0-b0bc-0024bed71c0a">
        <entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="
            http://activitystrea.ms/spec/1.0/">
          <id>0f72afbe-a9d4-11e0-b0bc-0024bed71c0a</id>
          <title>Alice posted a comment in the Coffee Talk
              conversation.</title>
          <summary>Alice posted a comment in the Coffee Talk
              conversation.</summary>
          <published>2011-07-01T13:00:00Z</published>
          <updated>2011-07-01T13:00:00Z</updated>
          <author>
            <name>Alice</name>
            <uri>acct:alice@example.com</uri>
            <activity:object-type>person</activity:object-type>
          </author>
          <activity:object>
            <id>0f72afbe-a9d4-11e0-b0bc-0024bed71c0a</id>
            <title>This is another nice comment.</title>
            <content type="text/html">This is another nice comment.</
                summary>
            <activity:object-type>comment</activity:object-type>
```

---

[5]XEP-0054: vcard-temp <https://xmpp.org/extensions/xep-0054.html>.

```
            </activity:object>
          </entry>
        </item>
      </items>
    </event>
</message>
```

4.5 Other Activity

Conversations MAY support submission of activity items other than comments. A client SHOULD first determine support for other item types before attempting to submit them. If a server does not support an item type, it should reject it:

Listing 11: Rejecting unsupported activity type

```
<iq type="error" from="comments.example.com" to="alice@example.com/1"
    id="3">
  <error type="modify">
    <bad-request xmlns="urn:ietf:params:xml:ns:xmpp-stanzas"/>
  </error>
</iq>
```

4.6 Mentions

If the server deems a submitted comment to be relevant to a user who is not subscribed to the activity node, it SHOULD send an unsolicited event to that user anyway. This way, users can "tag" or "mention" users not involved in a conversation, so that they may be notified about it. A comment is considered relevant to a user if one the following are true:

- The user is the one that submitted the comment. This is to allow a user service to automatically pick up on conversations that the user has commented in, without the user's client to have to explicitly inform the user service.

- The comment is a reply to one of the user's comments, or it affects one of the user's comments in some way (for example, modification by an admin).

- The comment body contains an HTML Microdata object of type "http://data-vocabulary.org/Person", where the itemid value is the user's account URI.

4.7 Deleted Comments

If a comment is deleted, it SHOULD remain in the past items of the node(s), but with its content cleared out and replaced with bogus author data and no activity:object. This change should also cause the comment item to be pushed out again to subscribers and relevant users. This way, entities that are tracking the conversation for changes can be informed of deletes.

Even after a network failure, the deleted items can be discovered by retrieving past items.

5 Implementation Notes

In order for the user information that gets saved with comments to not become stale over time, servers SHOULD have ways of refreshing this information by refetching user vcards.
To load a conversation intended for display with nesting, the following algorithm is RECOM-MENDED:

1. Let C be the desired number of total comments to display.

2. Request the C newest top-level comments (set parent_ids to an empty value).

3. Request the C newest comments at depth 1, by setting parent_ids to the list of comments in the previous request that had a reply count greater than 0.

4. Repeat previous step for every depth until the full depth has been traversed.

5. Truncate resulting tree to C items.

6 Security Considerations

As noted when handling comment submission above, the server MUST replace author information with that of the user performing the submission. This is essential to prevent author spoofing.
Care SHOULD be taken to prevent "mention spam." If the server determines a user is acting maliciously, then it MUST NOT send unsolicited events as a result of a submission. If a user service receives a mention event from a comment author that it has determined to be malicious, then it MUST NOT process the event further.

7 IANA Considerations

No interaction with the Internet Assigned Numbers Authority (IANA) [6] is required as a result of this document.

---

[6]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/>.

8 XMPP Registrar Considerations

8.1 Protocol Namespaces

This specification defines the following XML namespace:

- urn:xmpp:tmp:comments:0

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [7] shall add the foregoing namespace to the registry located at <https://xmpp.org/registrar/namespaces.html>, as described in Section 4 of XMPP Registrar Function (XEP-0053) [8].

8.2 Namespace Versioning

If the protocol defined in this specification undergoes a revision that is not fully backwards-compatible with an older version, the XMPP Registrar shall increment the protocol version number found at the end of the XML namespaces defined herein, as described in Section 4 of XEP-0053.

---

[7]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/>.

[8]XEP-0053: XMPP Registrar Function <https://xmpp.org/extensions/xep-0053.html>.