XEP-0376: Pubsub Account Management

Dave Cridland
mailto:dave@hellopando.com
xmpp:dwd@dave.cridland.net

2017-09-11
Version 0.2

Status	Type	Short Name
Deferred	Standards Track	pam

This specification describes a new model for handling remote pubsub services and a protocol for doing so.

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 **1**

2 User Stories **1**
- 2.1 Device Agility . . . 1
- 2.2 New Devices . . . 1
- 2.3 Offline Capability . . . 1
- 2.4 PEP . . . 1

3 Protocol **2**
- 3.1 Advertising Support . . . 2
  - 3.1.1 Clients . . . 2
  - 3.1.2 Servers . . . 2
- 3.2 Subscribing . . . 2
- 3.3 Unsubscribing . . . 4
- 3.4 Listing Subscriptions . . . 4
- 3.5 Auto Subscriptions . . . 5
- 3.6 Filtering . . . 5
- 3.7 Interaction with MAM . . . 5

4 Security Considerations **5**

5 XMPP Registrar Considerations **6**

6 IANA Considerations **6**

1 Introduction

The XMPP way is to have ”disposable”, or at least easily substituted, clients, maintaining long-term state on the server, and allowing it to be synchronized between clients. In particular, this can be seen on how the roster and presence fan-out operate - clients defer the operation of such things to the server, which manages the shared state and allows servers to access and manipulate it.
Historically, however, we have not done this for some more recently designed services, including Multi User Chat and PubSub. In both cases, different clients may be unaware of what chatrooms (etc) are joined (etc) by which other clients. This causes practical difficulty in seamlessly switching between devices and/or clients.

2 User Stories

2.1 Device Agility

- When a user subscribes to a publish-subscribe node (presumably via some higher-level abstraction), other online devices are aware of the new subscription immediately, and can choose to reflect the new subscription in their UI.
- Not all devices may be capable of handling the particular payload and/or service, and therefore should signal which payload and/or service types they support.
- The same capability as point 1 should be possible for unsubscribing.

2.2 New Devices

- When a user brings a new device online, it should be able to quickly learn all the user’s current subscriptions and present them to the user in its UI.

2.3 Offline Capability

- When the device is offline for an extended period (beyond XEP-0198 resumption capability), it needs to be able to obtain all the events it missed, including when the events occured.
- It should be able to tell which of these the user is unlikely to have seen on other devices.
- Further, it needs to be able to tell if new subscriptions have been added, or old ones removed.

2.4 PEP

- A one-way subscription to a user should still allow PEP.

- PEP should work the same way as now - users see filtered notifications about the things they care about.

3 Protocol

3.1 Advertising Support

3.1.1 Clients

Clients advertise support for this protocol via Service Discovery (XEP-0030) [1] using a Disco Feature of 'urn:xmpp:pam:0'. This is required for local servers to detect support.

3.1.2 Servers

Servers advertise this support via Service Discovery (XEP-0030) [2] on the user account (eg, <localpart@domain.tld>), using the same feature of 'urn:xmpp:pam:0'. This is used both by the local user and also remote pubsub services.

3.2 Subscribing

When a client wishes to subscribe to a node, either on the local server or remotely, using this protocol it does so by sending an <iq/> of type "set" to its own account, containing a pam element, which in turn has a service attribute (the target service jid) and a payload of a Publish-Subscribe (XEP-0060) [3] subscribe element (as described in Publish-Subscribe (XEP-0060) [4] §6.1). Example 32 from Publish-Subscribe (XEP-0060) [5] is thus performed in this protocol as follows:

Listing 1: Client subscribes to a node

```
<iq type='set' id='sub1'>
        <pam xmln='urn:xmpp:pam:0' jid='pubsub.shakespeare.lit'>
          <subscribe xmlns='http://jabber.org/protocol/pubsub'
              node='princely_musings'
              jid='francisco@denmark.lit'/>
        </pam>
      </iq>
```

Note that because the Publish-Subscribe (XEP-0060) [6] operation is intact within the pam element, local servers MAY interpret the operation, or MAY forward it verbatim. Note that

[1] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[2] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[3] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[4] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[5] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[6] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.

the client SHALL always use its own bare jid (eg, <localpart@domain.tld>) within a subscribe, servers MUST verify this.
Such a request SHALL cause the local server to send a traditional Publish-Subscribe (XEP-0060) [7] request, from the account bare jid, to the remote service.
When the remote service replies, the local server SHALL first notify all joined clients of the new subscription (described more in #sublist)...

Listing 2: Server notifies about new subscription

```
<message>
        <notify ver='aocolb' service='pubsub.shakespeare.lit' xmlns='
            urn:xmpp:pam:0'>
          <subscription xmlns='http://jabber.org/protocol/pubsub'
              node='princely_musings'
              jid='francisco@denmark.lit'
              subscription='subscribed'/>
       </notify>
     </message>
```

... and then MUST respond to the original <iq/>. Since the subscription has already been notified, this is an empty result <iq/>.
If the local server detects an error, it MUST NOT forward the request, and MUST respond with an <iq/> stanza of type error, which contains an error element which MAY be stamped with the local server as generator. Thus Example 34 from Publish-Subscribe (XEP-0060) [8] would be very similar:

Listing 3: An error generated remotely

```
<iq type='error' id='sub1'>
        <error type='modify' by='francisco@denmark.lit'>
          <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
          <invalid-jid xmlns='http://jabber.org/protocol/pubsub#errors
              '/>
        </error>
      </iq>
```

If the remote service rejects the subscription request, the local server simply forwards the response back as an <iq/> of type error, with the remote error copied through. The generator MUST be set to the remote service if missing. Thus Example 35 from Publish-Subscribe (XEP-0060) [9] might look as follows:

Listing 4: An error generated remotely

[7] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[8] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[9] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.

```
<iq type='error' id='sub1'>
    <error type='auth' by='pubsub.shakespeare.lit'>
      <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
      <presence-subscription-required xmlns='http://jabber.org/
         protocol/pubsub#errors'/>
    </error>
  </iq>
```

Clients MAY assume that if the generator is missing, the error is generated by the local server and not a remote service.

3.3 Unsubscribing

As above.

3.4 Listing Subscriptions

Clients obtain a current listing of the subscriptions, for example on initial connection, by sending a subscriptions request qualified by the pam namespace. If a client already has the opaque version identifier cached, it MAY include it within a "ver" attribute:

Listing 5: Client requests all current subscriptions

```
<iq type='get' id='subscriptions1'>
        <subscriptions xml='urn:xmpp:pam:0' ver='asdvcjkasdjb'>
      </iq>
```

The local server responds with either a response containing a subscription list (such as this, similar to Publish-Subscribe (XEP-0060) [10] Example 21):

Listing 6: Complete subscription list

```
<iq type='result' id='subscription1'>
        <subscriptions xml='urn:xmpp:pam:0' ver='kjlsadhfsd'>
          <subscription service='pubsub.shakespeare.lit' node='node1'
             jid='francisco@denmark.lit' subscription='subscribed'/>
          <subscription service='pubsub.marlowe.lit' node='node2' jid=
             'francisco@denmark.lit' subscription='subscribed'/>
          <subscription service='pubsub.marlowe.lit' node='node5' jid=
             'francisco@denmark.lit' subscription='unconfigured'/>
          <subscription service='pubsub.shakespeare.lit' node='node6'
             jid='francisco@denmark.lit' subscription='subscribed'
             subid='123-abc'/>
```

[10] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.

```
        <subscription service='pubsub.shakespeare.lit' node='node6'
            jid='francisco@denmark.lit' subscription='subscribed'
            subid='004-yyy'/>
      </subscriptions>
    </iq>
```

Alternately, a server MAY - if the client has supplied an opaque version identifier - send a sequence of <notify> elements followed by an empty <iq/> result.
Clients MAY persistently store the last "ver" attribute seen from either the <subscriptions> response or the last <notify>, whichever is later. This can then be used to minimize the volume of subscription data transferred during resync.

3.5 Auto Subscriptions

Servers need to subscribe to remote PEP services explicitly those nodes which are of interest. Interest needs to be detirmined by the client issuing a request; but this implies that servers would gradually acrue any node type which the user has had a capable client at any time. Perhaps timing out node types which have not been requested for over a certain period?
Clients can use +notify to handle auto-subscriptions between clients and their server.
Servers receiving +notify from accounts known to support this protocol ignore them.

3.6 Filtering

Clients filter subscriptions using a specific stanza (iq, probably), containing a list of node names. This can be used instead of the odler +notify (which is broadcast).

3.7 Interaction with MAM

We probably want to say that events are now archived by MAM, but this may imply that clients need to filter out such events (or explicitly include them). Maybe the mask above affects MAM queries?

4 Security Considerations

I have literally no idea. I don't think anything new is introduced that couldn't be discovered by traffic monitoring, although it collects and collates information that previously would not have been so readily available.

5 XMPP Registrar Considerations

On publication of this specification, the XMPP Registrar will dance a little jig to the tune of the traditional hornpipe with a tea-cosy upon his or her head.

6 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [11].

[11] 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/>.