XEP-0163: Personal Eventing Protocol

Peter Saint-Andre
mailto:peter@andyet.net
xmpp:stpeter@stpeter.im
https://stpeter.im/

Kevin Smith
mailto:kevin@kismith.co.uk
xmpp:kevin@doomsong.co.uk

2010-07-12
Version 1.2

Status	Type	Short Name
Draft	Standards Track	pep

This specification defines semantics for using the XMPP publish-subscribe protocol to broadcast state change events associated with an instant messaging and presence account. This profile of pubsub therefore enables a standard XMPP user account to function as a virtual pubsub service, easing the discovery of syndicated data and event notifications associated with such an account.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

1.1 Motivation

Personal eventing provides a way for a Jabber/XMPP user to send updates or "events" to other users, who are typically contacts in the user's roster. An event can be anything that a user wants to make known to other people, such as those described in User Geolocation (XEP-0080) [1], User Mood (XEP-0107) [2], User Activity (XEP-0108) [3], and User Tune (XEP-0118) [4]. While the XMPP Publish-Subscribe (XEP-0060) [5] extension ("pubsub") can be used to broadcast such events associated, the full pubsub protocol is often thought of as complicated and therefore has not been widely implemented. [6] To make publish-subscribe functionality more accessible (especially to instant messaging and presence applications that conform to XMPP IM [7]), this document defines a simplified subset of pubsub that can be followed by instant messaging client and server developers to more easily deploy personal eventing services across the Jabber/XMPP network. We label this subset "Personal Eventing Protocol" or PEP.
Note: Any use cases not described herein are described in XEP-0060. Also, this document does not show error flows related to the generic publish-subscribe use cases referenced herein, since they are exhaustively defined in XEP-0060. The reader is referred to XEP-0060 for all relevant protocol details related to the XMPP publish-subscribe extension. This document merely defines a "subset" or "profile" of XMPP publish-subscribe.

1.2 How It Works

This section provides a friendly introduction to personal eventing via pubsub (PEP).
Imagine that you are a Shakespearean character named Juliet and that you want to generate events about what music you're listening to, which anyone may see as long as they are authorized to see your online/offline presence (i.e., a pubsub access model of "presence").
We assume that you have three contacts with the following relationship to you:

1. benvolio@montague.lit, who has no subscription to your presence
2. nurse@capulet.lit, who has a bidirectional subscription to your presence and who is in your "Servants" roster group

[1] XEP-0080: User Geolocation <https://xmpp.org/extensions/xep-0080.html>.
[2] XEP-0107: User Mood <https://xmpp.org/extensions/xep-0107.html>.
[3] XEP-0108: User Activity <https://xmpp.org/extensions/xep-0108.html>.
[4] XEP-0118: User Tune <https://xmpp.org/extensions/xep-0118.html>.
[5] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[6] Instead, many "extended presence" formats are currently sent using the <presence/> stanza type; unfortunately, this overloads presence, results in unnecessary presence traffic, and does not provide fine-grained control over access. The use of publish-subscribe rather than presence is therefore preferable.
[7] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

3. romeo@montague.lit, who has a bidirectional subscription to your presence and who is in your "Friends" roster group

We also assume that your server (capulet.lit) supports PEP and that your client discovered that support when you logged in.
Now you start playing a song on your music playing software. Your client captures that "event" and publishes it to your server:

Listing 1: Publishing an event

```
<iq from='juliet@capulet.lit/balcony' type='set' id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
          <source>Music for "Love's Labors Lost" (Suite for small
              orchestra)</source>
          <title>Introduction (Allegro vigoroso)</title>
          <track>1</track>
        </tune>
      </item>
    </publish>
  </pubsub>
</iq>
```

Note the following about your publish request:

1. It is sent with no 'to' address (see Every Account a Pubsub Service).
2. It specifies a node of "http://jabber.org/protocol/tune" (see One Node per Namespace).

If all goes well (see Publishing Events), everyone who is interested in what you are listening to will receive notification of the event:

Listing 2: Interested parties receive event notifications

```
<message from='juliet@capulet.lit'
         to='romeo@montague.lit/orchard'
         type='headline'
         id='tunefoo1'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
```

```
          <source>Music for "Love's Labors Lost" (Suite for small
              orchestra)</source>
          <title>Introduction (Allegro vigoroso)</title>
          <track>1</track>
        </tune>
      </item>
    </items>
  </event>
</message>

<message from='juliet@capulet.lit'
         to='nurse@capulet.lit/chamber'
         type='headline'
         id='tunefoo2'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
          <source>Music for "Love's Labors Lost" (Suite for small
              orchestra)</source>
          <title>Introduction (Allegro vigoroso)</title>
          <track>1</track>
        </tune>
      </item>
    </items>
  </event>
</message>
```

Because PEP services must send notifications to the account owner, you too receive the notification at each of your resources (here "balcony" and "chamber").

Listing 3: Publisher receives event notification

```
<message from='juliet@capulet.lit'
         to='juliet@capulet.lit/balcony'
         type='headline'
         id='tunefoo3'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
          <source>Music for "Love's Labors Lost" (Suite for small
              orchestra)</source>
          <title>Introduction (Allegro vigoroso)</title>
          <track>1</track>
```

```
        </tune>
      </item>
    </items>
  </event>
</message>

<message from='juliet@capulet.lit'
         to='juliet@capulet.lit/chamber'
         type='headline'
         id='tunefoo4'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
          <source>Music for "Love's Labors Lost" (Suite for small
              orchestra)</source>
          <title>Introduction (Allegro vigoroso)</title>
          <track>1</track>
        </tune>
      </item>
    </items>
  </event>
</message>
```

But how do Romeo and the Nurse tell your server that they are interested in knowing what you're listening to? In generic pubsub they typically need to explicitly subscribe to your "http://jabber.org/protocol/tune" node. [8] But PEP services support two special features:

1. "auto-subscribe" -- because they are subscribed to your presence, they automatically receive your events (see Use Presence).
2. "filtered-notification" -- they can include some special flags in their Entity Capabilities (XEP-0115) [9] information to specify which event types (payloads) they want to receive (see Filtered Notifications).

Listing 4: Romeo sends presence with caps

```
<presence from='romeo@montague.lit/orchard'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://www.chatopus.com'
     ver='zHyEOgxTrkpSdGcQKH8EFPLsriY='/>
</presence>
```

[8] That may still be necessary for open access model nodes in PEP if another user does not send you presence, such as benvolio@montague.lit in our scenario.

[9] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

Your server knows to send tune information to Romeo because when the server unpacks the value of the 'ver' attribute ("054H4A7280JuT6+IroVYxgCAjZo=") in accordance with XEP-0115, it discovers that Romeo's client advertises a service discovery feature of "http://jabber.org/protocol/tune+notify", where the "+notify" suffix indicates interest in receiving notifications related to the protocol that precedes the suffix. The server can verify this support if needed by sending a service discovery request to Romeo's full JID, where the response would be as follows:

Listing 5: Disco#info result from extension

```
<iq from='romeo@montague.lit/orchard'
    to='juliet@capulet.lit'
    type='result'
    id='disco123'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='client' name='Exodus␣0.9.1' type='pc'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='http://jabber.org/protocol/geoloc'/>
    <feature var='http://jabber.org/protocol/geoloc+notify'/>
    <feature var='http://jabber.org/protocol/tune'/>
    <feature var='http://jabber.org/protocol/tune+notify'/>
  </query>
</iq>
```

Naturally your server doesn't need to send out a disco#info request every time, since it will quickly create a large cache of 'ver' values.
So that's the general idea.

2 Concepts and Approach

Personal eventing via pubsub ("PEP") is based on the following principles:

1. Every account a pubsub service.
2. One publisher per node.
3. Use presence.
4. Filter notifications based on expressed interest.
5. Smart defaults.

These principles are described more fully below.

2.1 Every Account a Pubsub Service

When a user creates an account (or has an account provisioned) at a Jabber/XMPP server that supports PEP, the server associates a virtual pubsub service with the account. This greatly simplifies the task of discovering the account owner's personal pubsub nodes, since the root pubsub node simply is the account owner's bare JID (<localpart@domain.tld> or <domain.tld>). This assumption also simplifies publishing and subscribing.

2.2 One Publisher Per Node

There is no need for multiple publishers to a PEP service, since by definition the service generates information associated with only one entity. The owner-publisher for every node is the bare JID of the account owner.

2.3 Use Presence

Although generic publish-subscribe services do not necessarily have access to presence information about subscribers, PEP services are integrated with presence in the following ways:

- Each messaging and presence account simply *is* a virtual publish-subscribe service.
- The default access model is "presence".
- A contact's subscription to an account owner's personal eventing data is automatically created because the contact has an XMPP presence subscription (the "auto-subscribe" feature).
- Services take account of subscriber presence in the generation of notifications. [10]
- A service automatically sends notifications to all of the account owner's connected resources (subject to notification filtering).

These uses of presence simplify the task of developing compliant clients (cf. XMPP Design Guidelines (XEP-0134) [11]).
Note: It is strongly NOT RECOMMENDED to use directed presence with Entity Capabilities data that differs from the data included in broadcast presence for the purpose of establishing implicit PEP subscriptions to another entity, because the directed presence information will be overwritten by any subsequent presence broadcast.

[10] This works only if the subscription state is "both" (see RFC 3921).
[11] XEP-0134: XMPP Design Guidelines <https://xmpp.org/extensions/xep-0134.html>.

2.4 Filtered Notifications

By default, the existence of an XMPP presence subscription is used to establish a PEP subscription to the account owner's personal eventing data. In order to filter which notifications are sent by the PEP service, the contact's client includes extended Entity Capabilities (XEP-0115) [12] information in the presence notifications it sends to the account owner. Because the PEP service supports the "filtered-notifications" feature, it sends only those notifications that match the contact's expressed notification preferences.

2.5 Smart Defaults

Most pubsub configuration options and metadata are not needed for personal eventing. Instead, PEP services offer smart defaults to simplify node creation and management.

3 Publishing Events

An account owner publishes an item to a node by following the protocol specified in XEP-0060:

Listing 6: Account owner publishes item

```
<iq from='juliet@capulet.lit/balcony' type='set' id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
          <source>Music for "Love's Labors Lost" (Suite for small
              orchestra)</source>
          <title>Introduction (Allegro vigoroso)</title>
          <track>1</track>
        </tune>
      </item>
    </publish>
  </pubsub>
</iq>
```

If the node does not already exist, the PEP service MUST create the node. This "auto-create" feature (defined in XEP-0060) MUST be supported by a PEP service. (Naturally, the account owner's client MAY follow the node creation use case specified in XEP-0060 before attempting to publish an item.)
A PEP service SHOULD also support the "publish-options" feature defined in XEP-0060.
If the publication logic dictates that event notifications shall be sent, the account owner's

[12] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

server generates notifications and sends them to all appropriate entities as described in the Receiving Event Notifications section of this document, as well as to any of the account owner's available resources.
Note: PEP ties the receipt of PEP notifications to the subscriber's presence, but does not tie the generation of PEP notifications to the publisher's presence. If the publisher wishes to stop generating PEP events (or to generate an "empty" event as can be done for some PEP payloads) before ending its presence session, the publisher MUST direct its client to do so and MUST NOT depend on the PEP service to automatically "zero out" its PEP information when the PEP service receives unavailable presence from the publisher.

4 Receiving Event Notifications

An entity shall receive event notifications if:

1. The node has an open access model and the entity has explicitly or implicitly subscribed to the node as explained in XEP-0060.
2. The entity shares presence with the account owner (see Presence Sharing), is authorized to receive events from the node in accordance with the node access model (see XEP-0060), and advertises an interest in the payload type (see Notification Filtering).
3. The entity is the account owner itself, in which case the PEP service shall send notifications to all of the account owner's available resources (subject to notification filtering).

4.1 Automatic Subscriptions

A PEP service MUST support the "auto-subscribe" feature defined in Section 9.1 of XEP-0060. This implies that when a user has an XMPP presence subscription to the account owner's presence, the user automatically also has the right to subscribe to any of the account owner's PEP nodes (if the access model is the default of "presence") and to retrieve items from such PEP nodes.

4.2 Notification Filtering

A PEP service MUST support the "filtered-notifications" feature defined in Section 9.2 of XEP-0060. This implies that when an automatic subscriber can specify which event payloads it wants to receive by including appropriate feature bundles in the XEP-0115 information it broadcasts.

4.3 Generating Notifications

4.3.1 Addressing

1. The server MUST set the 'from' address on the notification to the bare JID (<localpart@domain.tld> or <domain.tld>) of the account owner (in these examples, "juliet@capulet.lit").

2. Any errors generated by the recipient or the recipient's server in relation to the notification MUST be directed to the JID of the 'from' address on the notification (i.e., the bare JID) so that bounce processing can be handled by the PEP service rather than by the publishing client.

3. When sending notifications to an entity that has a presence subscription to the account owner, the server SHOULD include an Extended Stanza Addressing (XEP-0033) [13] "replyto" extension specifying the publishing resource (in this example, "juliet@capulet.lit/balcony"); this enables the subscriber's client to differentiate between information received from each of the account owner's resources (for example, different resources may be in different places and therefore may need to specify distinct geolocation data). However, a server MUST NOT include the "replyto" address when sending a notification to an entity that does not have a presence subscription to the account owner.

4. If the PEP service has presence information about the intended recipient, it SHOULD direct the notification(s) to the full JID(s) of the recipient's (<localpart@domain.tld/resource> or <domain.tld/resource>); if the PEP service does not have presence information about a subscriber, it MUST address the notification to the subscriber's bare JID (<localpart@domain.tld> or <domain.tld>).

4.3.2 Number of Notifications

1. If an entity subscribed using a full JID (<localpart@domain.tld/resource> or <domain.tld/resource>) or a bare domain identifier <domain.tld>, a PEP service MUST send one notification only, addressed to the subscribed JID.

2. If a subscriber subscribed using a bare JID <localpart@domain.tld> and a PEP service does not have appropriate presence information about the subscriber, a PEP service MUST send at most one notification, addressed to the bare JID <localpart@domain.tld> of the subscriber, and MAY choose not to send any notification. (By "appropriate presence information" is meant an available presence stanza with XEP-0115 data that

[13] XEP-0033: Extended Stanza Addressing <https://xmpp.org/extensions/xep-0033.html>.

indicates interest in the relevant data format.)

3. If a subscriber subscribed using a bare JID <localpart@domain.tld> and a PEP service has appropriate presence information about the subscriber, the PEP service MUST send one notification to the full JID (<localpart@domain.tld/resource> or <domain.tld/resource>) of each of the subscriber's available resources that have included XEP-0115 information indicating an interest in the data format.

4.3.3 When to Generate Notifications

1. When an account owner publishes an item to a node, a PEP service MUST generate a notification and send it to all appropriate subscribers (where the number of notifications is determined by the foregoing rules).

2. When a PEP service receives initial presence [14] from a subscriber's resource including XEP-0115 information that indicates an interest in the data format, it MUST generate a notification containing at least the last published item for that node and send it to the newly-available resource; see below under Sending the Last Published Item.

3. As an exception to the foregoing MUST rules, a PEP service MUST NOT send notifications to a subscriber if the user has blocked the subscriber from receiving the kind of stanza used for notifications (typically message stanzas) by means of communications blocking as specified in Privacy Lists (XEP-0016) [15] or Blocking Command (XEP-0191) [16].

4.3.4 Sending the Last Published Item

As mentioned, a PEP service MUST send the last published item to all new subscribers and to all newly-available resources for each subscriber, including the account owner itself. (That is, the default value of the "pubsub#send_last_published_item" node configuration field must be "on_sub_and_presence"; this behavior essentially mimics the functionality of presence as defined in XMPP IM.) When processing a new subscription, the service MAY send not only the last published item but instead all items that are currently associated with the node (i.e., up to the maximum number of items at the node, which might be one if the node is a "singleton node" as described in XEP-0060). If the service has knowledge about the datetime that a subscriber's newly-available resource last received updated information from the node (e.g.,

[14] By "initial presence" is meant a presence stanza with no 'type' attribute that the PEP service receives after the subscriber was previously unavailable; any subsequent presence stanza with no 'type' attribute that the PEP service receives after the initial presence notification but before the subscriber against goes offline MUST NOT trigger sending of a new pubsub notification.
[15] XEP-0016: Privacy Lists <https://xmpp.org/extensions/xep-0016.html>.
[16] XEP-0191: Blocking Command <https://xmpp.org/extensions/xep-0191.html>.

as described in Last Activity in Presence (XEP-0256) [17]) then it MAY also send more items that only the last published item to the newly-available resource.
Note: The "on_sub_and_presence" setting relates to the *subscriber's* presence, not the publisher's presence.

Listing 7: Subscriber sends presence from newly-available resource

```
<presence from='romeo@montague.lit/orchard'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://www.chatopus.com'
     ver='zHyEOgxTrkpSdGcQKH8EFPLsriY='/>
</presence>
```

Listing 8: Subscriber's server sends presence from newly-available resource to publisher's bare JID (i.e., PEP service)

```
<presence from='romeo@montague.lit/orchard' to='juliet@capulet.lit'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://www.chatopus.com'
     ver='zHyEOgxTrkpSdGcQKH8EFPLsriY='/>
</presence>
```

Listing 9: PEP service sends last published item to newly-available resource

```
<message from='juliet@capulet.lit'
         to='romeo@montague.lit/orchard'
         type='headline'
         id='foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
          <source>Music for "Love's Labors Lost" (Suite for small
             orchestra)</source>
          <title>Introduction (Allegro vigoroso)</title>
          <track>1</track>
        </tune>
      </item>
    </items>
  </event>
  <delay xmlns='urn:xmpp:delay' stamp='2003-12-13T23:58:37Z'/>
</message>
```

[17] XEP-0256: Last Activity in Presence <https://xmpp.org/extensions/xep-0256.html>.

5 Recommended Defaults

A PEP service MUST:

- Support the node discovery, node creation, node deletion, publish item, subscribe, unsubscribe, and item retrieval use cases specified in XEP-0060.
- Support the "auto-create", "auto-subscribe", and "filtered-notifications" features.
- Support the "owner" and "subscriber" affiliations.
- Support the "presence" access model and set it to the default.
- Support the "open", "roster", and "whitelist" access models.
- Treat the account owner's bare JID (<localpart@domain.tld> or <domain.tld>) as a collection node (i.e., as the root collection node for the account's virtual pubsub service).
- Default the 'deliver_notifications' configuration option to true (i.e., deliver payloads by default).
- Default the 'send_last_published_item' configuration option to on_sub_and_presence (i.e., send the last published item on subscription and on receipt of presence). [18]

A PEP service MAY support other use cases, affiliations, access models, and features, but such support is OPTIONAL.

6 Determining Support

6.1 Account Owner Service Discovery

Naturally, before an account owner attempts to complete any PEP use cases, its client SHOULD determine whether the account owner's server supports PEP; to do so, it MUST send a Service Discovery (XEP-0030) [19] information request to its own bare JID:

Listing 10: Account owner queries server regarding protocol support

```
<iq from='juliet@capulet.lit/balcony'
    to='juliet@capulet.lit'
    id='disco1'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

[18] Because subscriptions are implicit in PEP rather than explicit as in generic pubsub, the on_sub_and_presence setting effectively means sending on presence.

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

If the account owner's server supports PEP and the account is provisioned for PEP, the server MUST return an identity of "pubsub/pep" on behalf of the account (as well as a list of the namespaces and other features it supports, including all supported XEP-0060 features):

Listing 11: Server communicates protocol support

```
<iq from='juliet@capulet.lit'
    to='juliet@capulet.lit/balcony'
    id='disco1'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='account' type='registered'/>
    <identity category='pubsub' type='pep'/>
    <feature var='http://jabber.org/protocol/pubsub#access-presence'/>
    <feature var='http://jabber.org/protocol/pubsub#auto-create'/>
    <feature var='http://jabber.org/protocol/pubsub#auto-subscribe'/>
    <feature var='http://jabber.org/protocol/pubsub#config-node'/>
    <feature var='http://jabber.org/protocol/pubsub#create-and-
       configure'/>
    <feature var='http://jabber.org/protocol/pubsub#create-nodes'/>
    <feature var='http://jabber.org/protocol/pubsub#filtered-
       notifications'/>
    <feature var='http://jabber.org/protocol/pubsub#persistent-items'/
       >
    <feature var='http://jabber.org/protocol/pubsub#publish'/>
    <feature var='http://jabber.org/protocol/pubsub#retrieve-items'/>
    <feature var='http://jabber.org/protocol/pubsub#subscribe'/>
    ...
  </query>
</iq>
```

6.2 Contact Service Discovery

A contact MAY send service discovery requests to the account owner's bare JID (<local-part@domain.tld> or <domain.tld>). If the contact already has a subscription to the account owner's presence, this is not necessary in order to receive notifications from the account owner via personal eventing. However, a user without a presence subscription needs to do so in order to discover if the account owner is a virtual pubsub service and to discover the account owner's eventing nodes. The relevant protocol flows are demonstrated in XEP-0060. Note: When returning disco#items results, the account owner's server MUST check the access model for each of the account owner's PEP nodes and MUST return as service discovery items only those nodes to which the contact is allowed to subscribe or from which the contact is allowed to retrieve items without first subscribing.

7 Implementation Notes

7.1 Cancelling Subscriptions

In order to ensure appropriate access to information published at nodes of type ”presence” and ”roster”, a PEP service MUST re-calculate access controls when:

1. A presence subscription state changes (e.g., when a subscription request is approved).
2. A roster item is modified (e.g., when the item is moved to a new roster group).

If the modification results in a loss of access, the service MUST cancel the entity’s subscription. In addition, the service MAY send a message to the (former) subscriber informing it of the cancellation (for information about the format of messages sent to notify subscribers of subscription cancellation, see the ”Notification of Subscription Denial or Cancellation” section of XEP-0060).

7.2 One Node Per Namespace

An earlier version of this document specified that there could be only one publish-subscribe node associated with any given payload type (XML namespace) for the account owner (e.g., there could be only one pubsub node for geolocation events, one node for tune events, and one node for mood events, etc.). However, this rule is now considered overly restrictive because some data formats can be used to encapsulate many different kinds of information; the usual example is Atom as defined in RFC 4287 [20], for which many extensions exist. Therefore, this document now does not specify that there is a one-to-one relationship between NodeIDs and payload namespaces.
A specification that defines a given payload format for use in PEP MUST specify whether there shall be only one node per namespace, or whether multiple NodeIDs for the same namespace are allowable.

8 Security Considerations

A PEP service MAY enforce additional privacy and security policies when determining whether an entity is allowed to subscribe to a node or retrieve items from a node; however, any such policies shall be considered specific to an implementation or deployment and are out of scope for this document.

[20] RFC 4287: The Atom Syndication Format <http://tools.ietf.org/html/rfc4287>.

9 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [21].

10 XMPP Registrar Considerations

10.1 Service Discovery Category/Type

The XMPP Registrar [22] includes a category of "pubsub" in its registry of Service Discovery identities (see <https://xmpp.org/registrar/disco-categories.html>); as a result of this document, the Registrar includes a type of "pep" to that category.
The registry submission is as follows:

```
<category>
  <name>pubsub</name>
  <type>
    <name>pep</name>
    <desc>
      A personal eventing service that supports the
      publish-subscribe subset defined in XEP-0163.
    </desc>
    <doc>XEP-0163</doc>
  </type>
</category>
```

11 XML Schema

Because PEP simply reuses the protocol specified in XEP-0060, a separate schema is not needed.

12 Acknowledgements

The authors wish to thank the participants in the XMPP Interoperability Testing Event held July 24 and 25, 2006, who provided valuable feedback that resulted in radical simplification of the protocol.

[21] 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/>.

[22] 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/>.

Thanks also to the many members of the standards@xmpp.org discussion list who patiently suffered through seemingly endless discussion of the auto-create and publish-and-configure features.