XEP-0021: Jabber Event Notification Service (ENS)

Robert Norris
mailto:rob@cataclysm.cx
xmpp:rob@cataclysm.cx

2003-04-22
Version 0.2

Status	Type	Short Name
Retracted	Standards Track	None

A generic publish-and-subscribe service for Jabber.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction **1**

2 Definitions **1**
- 2.1 Event . . . 1
- 2.2 Subscriber . . . 2
- 2.3 Publisher . . . 2

3 Subscriber protocol **3**
- 3.1 Subscription request . . . 3
  - 3.1.1 Option: <auth-info/> . . . 3
  - 3.1.2 Option: <reliable/> . . . 3
- 3.2 Subscription response . . . 4
- 3.3 Unsubscription request . . . 4
- 3.4 Unsubscription response . . . 4
- 3.5 Publish . . . 5
- 3.6 Publish response (reliable subscriptions) . . . 5

4 Publisher protocol **6**
- 4.1 Publish . . . 6
- 4.2 Publish response . . . 6
- 4.3 Authorisation request . . . 6
- 4.4 Authorisation response . . . 7

5 Errors **8**

6 Considerations **8**
- 6.1 ENS discovery . . . 8
- 6.2 How reliable should <reliable/> be? . . . 9
- 6.3 Longevity of subscriptions . . . 9
- 6.4 Replacing <presence/> with the ENS? . . . 9

1 Introduction

The Jabber Event Notification Service (ENS) acts as a dispatcher and may be used by applications as a central point of collection for certain types of events that are of interest to them. Examples of events include:

- User has logged in
- A new message has arrived
- User's avatar has changed
- The coffee machine is empty

In Jabber, the role of the ENS has traditionally been filled by overloading the <presence/> packet type. However, this method was never designed to be used as a general publish-and-subscribe mechanism, and so has the following problems:

- Dispatching of <presence/> packets is performed by the JSM (Jabber Session Manager), and so is not easily usable by components and other entities that don't connect via a client manager (c2s, CCM).
- An entity cannot subscribe to the presence of a specific resource of another entity, only to any presence from that entity. This lack of granularity makes its difficult to use <presence/> in situations where large chunks of data must be dispatched to subscribers (eg avatars).

The protocol consists of two parts - the subscriber-to-ENS protocol, and the publisher-to-ENS protocol. Since there is no direct interaction between a publisher and a subscriber, it makes sense to seperate the two parts of the protocol.
The protocol operates in the 'http://xml.cataclysm.cx/jabber/ens/' namespace.
A reference implementation was formerly available at http://cataclysm.cx/jabber/ens.html.

2 Definitions

Before we begin describing the protocol, is it necessary to define some terms, including the function and responsibilties of the entities that communicate with the ENS.

2.1 Event

An event can be defined as a change to the value of one or more properties of a resource.
In the ENS, an event type is referred to by a JID (Jabber IDentifier), including the JID resource. For example, consider a hypothetical publisher that is linked to an IMAP message store.

It might notify the ENS of the fact the a message has arrived, deleted, or filed, using the following JIDs:

- rob@imap.cataclysm.cx/NewMessage
- joe@imap.cataclysm.cx/DeletedMessage
- jim@imap.cataclysm.cx/FiledMessage

Alternatively, an end-user client that wanted to notify the ENS when its avatar changes might do so using a JID like "rob@cataclysm.cx/avatar"

2.2 Subscriber

A subscriber is a Jabber entity that receives notifications about events. Usually, a subscriber will be an end-user client, but it may be any Jabber entity.
As the name suggests, a subscriber can subscribe and unsubscribe to various events via the ENS. When it subscribes, the publisher responsible for the event it is subscribing to will be asked by the ENS to authorise the subscription request. To facilitate this, the subscriber may provide an XML fragment containing information that the publisher can use to authorise it. The use of this fragment is application specific.
Once subscribed to an event, the subscriber will receive any notifications that the publisher sends about that event.

2.3 Publisher

A publisher is the Jabber entity responsible for actually sending event notifications to the ENS. A notification contains the event type JID of the event that occured, and an optional "payload" XML fragment, that is passed untouched by the ENS to the subscriber. The contents of this payload is application-specific and can be used to provide detailed information about the event to the subscriber. For example, in the case of the NewMessage event above, the payload might contain the contents of the To:, From: and Subject: headers of the message.
Additionally, the publisher is responsible for deciding who may subscribe to events it publishes. When the ENS receives a subscription request, it will ask the publisher to decide whether or not the subscriber may subscribe to a particular event. This authorisation request may also contain an XML fragment from the subscriber containing information that may be used for authorisation.

3 Subscriber protocol

3.1 Subscription request

To subscribe to a particular event, the subscriber sends a packet like of the following form to the ENS:

Listing 1: Subscription request

```
<iq id='sub1' type='set' from='subscriber-jid' to='ens-jid'>
  <subscribe xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event-
      jid'/>
</iq>
```

3.1.1 Option: <auth-info/>

The subscriber may include an <auth-info/> XML fragment containing some (application-specific) information that the publisher can use to authorise it:

Listing 2: Subscription request with authorisation information

```
<iq id='sub2' type='set' from='subscriber-jid' to='ens-jid'>
  <subscribe xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event-
      jid'>
    <auth-info xmlns='jabber:iq:auth'>password</auth-info>
  </subscribe>
</iq>
```

3.1.2 Option: <reliable/>

If it wishes, the subscriber may request a "reliable" subscription. This option guarantees that the subscriber will receive all notifications about this event (as far as the Jabber architecture guarantees delivery). This changes the semantics of the subscriber publish protocol - see section 3.6 for more details.

Listing 3: Reliable subscription request

```
<iq id='sub2' type='set' from='subscriber-jid' to='ens-jid'>
  <subscribe xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event-
      jid'>
    <reliable/>
  </subscribe>
</iq>
```

3.2 Subscription response

Once subscribed, the ENS will return a packet of the following form to the subscriber:

Listing 4: Successful subscription response

```
<iq id='sub1' type='result' from='ens-jid' to='subscriber-jid'>
  <subscribed xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event-
      jid'/>
</iq>
```

If an error occured during subscription (such as the publisher not allowing the subscriber to subscribe), an error packet will be returned to the subscriber:

Listing 5: Failed subscription response

```
<iq id='sub1' type='error' from='ens-jid' to='subscriber-jid'>
  <subscribe xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event-
      jid'/>
  <error code='401'>Unauthorized</error>
</iq>
```

The actual error fragment in the packet is a direct copy of the one returned by the publisher when it fails the authorisation request from the ENS. If the publisher does not provide one, error code 503 (Service Unavailable) will be returned instead. If the publisher does not respond to the authorisation request (after an implementation-specific timeout), error code (Remote Server Timeout) will be returned.

3.3 Unsubscription request

To unsubscribe from a particular event, the subscriber sends a packet like of the following form to the ENS:

Listing 6: Unsubscription request

```
<iq id='unsub1' type='set' from='subscriber-jid' to='ens-jid'>
  <unsubscribe xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event-
      jid'/>
</iq>
```

3.4 Unsubscription response

Once unsubscribed, the ENS will return a packet of the following form to the subscriber:

Listing 7: Unsubscription response

```
<iq id='unsub1' type='result' from='ens-jid' to='subscriber-jid'>
  <unsubscribed xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event
      -jid'/>
</iq>
```

No further notifications for the event will be received.

3.5 Publish

When a publisher publishes a notification to the ENS, the ENS will forward the notification to any subscribers for that event. A notification sent to a subscriber takes the following form:

Listing 8: Event notification (publish)

```
<iq id='enspub1' type='set' from='ens-jid' to='subscriber-jid'>
  <publish xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event-jid'
      />
</iq>
```

A notification may also contain a (application-specific) "payload" XML fragment:

Listing 9: Event notification (publish) with payload

```
<iq id='enspub2' type='set' from='ens-jid' to='subscriber-jid'>
  <publish xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='event-jid'
      >
    <xml-payload/>
  </publish>
</iq>
```

Section 4.1 has more information about the payload.
If the subscriber responds to the notification with an error, the subscriber will be automatically unsubscribed from the event by the ENS.

3.6 Publish response (reliable subscriptions)

If the <reliable/> option was specified when the subscriber subscribed to the event, then the subscriber is expected to acknowledge a notification with a packet of the following form:

Listing 10: Event notification (publish) response

```
<iq id='enspub1' type='result' from='subscriber-jid' to='ens-jid'>
  <published xmlns='http://xml.cataclysm.cx/jabber/ens/'/>
</iq>
```

If the subscriber does not respond, or responds with an error, the notification will be resent by the ENS after a (implementation-specific) timeout.

4 Publisher protocol

4.1 Publish

To publish a notification, the publisher sends a packet of the following form to the ENS:

Listing 11: Event notification (publish)

```
<iq id='pub1' type='set' from='event-jid' to='ens-jid'>
  <publish xmlns='http://xml.cataclysm.cx/jabber/ens/'/>
</iq>
```

A notification may also contain a (application-specific) "payload" XML fragment:

Listing 12: Event notification (publish) with payload

```
<iq id='pub1' type='set' from='event-jid' to='ens-jid'>
  <publish xmlns='http://xml.cataclysm.cx/jabber/ens/'>
    <xml-payload/>
  </publish>
</iq>
```

The payload can be any well-formed XML data. Everything inside the <publish/> tag will be forwarded untouched to the subscriber.

4.2 Publish response

Once the ENS has dispatched the notification to subscribers, it will return a packet of the following form to the publisher:

Listing 13: Event notification (publish) response

```
<iq id='pub1' type='result' from='ens-jid' to='event-jid'>
  <published xmlns='http://xml.cataclysm.cx/jabber/ens/'/>
</iq>
```

4.3 Authorisation request

A publisher is required to approve subcription requests. When a subscriber attempts to subscribe to an event, the publisher will receive a packet of the following form from the ENS:

Listing 14: Authorisation request

```
<iq id='ensauth1' type='get' from='ens-jid' to='event-jid'>
  <authorise xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='
      subscriber-jid'/>
</iq>
```

The subscriber may include an <auth-info/> XML fragment containing some (application-specific) information that the publisher can use to authorise it:

Listing 15: Authorisation request with authorisation information

```
<iq id='ensauth1' type='get' from='ens-jid' to='event-jid'>
  <authorise xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='
      subscriber-jid'>
    <auth-info xmlns='jabber:iq:auth'>password</auth-info>
  </authorise>
</iq>
```

4.4 Authorisation response

To signal to the ENS that a subscriber should be allowed to subscribe, the publisher should return a packet of the following form:

Listing 16: Successful authorisation response

```
<iq id='ensauth1' type='result' from='event-jid' to='ens-jid'>
  <authorised xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='
      subscriber-jid'/>
</iq>
```

To deny the subscription, the publisher should return an error packet, containing the appropriate <error/> fragment:

Listing 17: Unsuccessful authorisation response

```
<iq id='ensauth1' type='error' from='event-jid' to='ens-jid'>
  <authorise xmlns='http://xml.cataclysm.cx/jabber/ens/' jid='
      subscriber-jid'/>
  <error code='401'>Unauthorized</error>
</iq>
```

The <error/> fragment will be copied untouched into the error response sent back to the subscriber. See section 3.2 for more details.

5 Errors

The ENS will respond with error code 400 (Bad Request) any time it receives a request that it cannot understand (a malformed packet, for example).
Other errors may occur, and they are described in the appropriate sections of the above protocol specification.

6 Considerations

The following items should be discussed before this proposal is approved by the Council:

6.1 ENS discovery

The ENS, as described, works well. However, no provisions are made for a subscriber to find out which ENS a publisher is publishing to. It does a subscriber no good to subscribe to an event on the wrong ENS - the subscription would succeed (assuming the publisher allows it), but no notifications would ever be received.
There are several potential solutions, each with their problems:

- Leave it to the subscriber to find the appropriate ENS outside of the ENS framework itself. This might be via a browse to the publisher, or maybe just entering it into their client's configuration. Obviously, this is very application-specific.
- Force the publisher to publish to multiple ENSs, as necessary. This would require additions to the protocol (to tell the publisher of new ENSs), and would require the publisher to maintain a list of JIDs it needs to publish to. This solution is pointless - if the publisher is going to publish to multiple JIDs and maintain its own list, it might as well publish direct to subscribers.
- Have some sort of ENS-to-ENS protocol, and have ENSs proxy publishes for other ENSs. This does not fix the problem, it just moves it away from the subscriber and into the ENS. An ENS will still need to find out which ENS the publisher is publishing to.
- Integrate ENS into the session manager. This leaves us with a glorified presence system, and makes the ENS basically unusable by non-session-manager-based server components.

This problem may be outside of the scope of this specification.

6.2 How reliable should <reliable/> be?

Currently, if a subscriber obtains a reliable subscription, and then disappears from the network (as an end-user client might), the ENS will continue to send notifications to it (ignoring errors) until it unsubscribes. If the subscriber never comes back, then ENS will send notifications forever (in theory).
At what point should even <reliable/> have its limits? Should we unsubscribe a reliable subscriber who bounces (ie. the Jabber server failed to deliver) more than, say, 10 publishes? Or maybe they should be unsubscribed if they do not respond (as in section 3.6) to anything for, say, 10 minutes?
<reliable/> is an interesting idea, but it may be that it is too problematic for its own good.

6.3 Longevity of subscriptions

The topic of reliable subscriptions raises the question as to how long a subscription lasts. Should a subscription last forever (like <presence/> does), even across restarts of the server? If end-user clients are to be subscribers, then under this scheme the ENS would have to subscribe to presence, so as to know when the client has disconnected. Since presence is a function of the session manager, this could have the effect of making the ENS less generic that we may like.

6.4 Replacing <presence/> with the ENS?

As I see it, basic presence will always be a function of the session manager, for the following reasons:

- No ENS discovery problems
- Subscriptions are maintained across sessions
- In very widespread use (ie. everywhere)

I think the places where the ENS can boom will be in applications like avatars, and genuine event-based systems (like the kind described above - ”you have new mail”, ”someone has scheduled a meeting with you in your online calendar”, etc).