From buddycloud wiki

(Redirected from xMPP XEP)

buddycloud sequence diagram XMPP API

This is the working area for the buddycloud channels XEP.

Below is all the content that will be considered to go into the XEP. We are attempting to only put only the necessary protocol bits into the XEP.

Contents

Introduction

buddycloud attempts to design a decentralised social product that solves users needs in a beautiful and efficient way.

buddycloud builds on XMPP's native federation and asynchronous messaging to provide a decentralised sharing and communication network. The buddycloud ecosystem contains multiple [XMPP] components (for example: location butler, media server and directory) that together form a cohesive product and enable developers to build great products for their users.

Glossary

buddycloud channel
: set of content-specific pubsub nodes per user (posts, geoloc, mood)

Home Server
: hosts your channel nodes and is your inbox. The home server can do things on a behalf of user (like subscribing to a whitelisted channel))

Remote Server
: hosts channels nodes of another user

Follower
: a channel subscriber

Moderator
: a channel follower with capabilities to also approve followers to topic channels and retract posts

Producer
: the owner of a channel with the ability to add and remove moderators

personal channel
: a channel about a user and named after their jid (e.g. hag66@shakespeare.lit)

topic channel
: a channel based on a topic and carrying slightly different business logic (e.g. cauldron-recipees@shakespear.lit)

Associated documents

- Place Management
- Channel Directory

buddycloud Logic

By prescribing sensible behavior that a server can enforce we avoid support issues on clients. "I've lost publishing rights to my own channel" or "I can't view a friends channel even though I am a moderator in that channel". Pub-sub is a great backend. A good user service built on it will prescribe sensible business logic that fits with a users mental model of what channels provide. We do that by adding a bit of business logic to the buddycloud server:

- channels nodes are owned by their jid. For example channeluser@example.com owns channel.example.com, node /user/channeluser@example.com/posts
- the buddycloud server should maintain similar affiliations across .../posts, .../geo/current, .../geo/previous, .../geo/future and .../mood ie, if you follow a user, you also get to see their status.
- only moderators and producers should be able to see channel outcasts (don't glorify bad behavior)
- use predictable channel addresses for web users.
- overview of channel services: http://example.com
- individual user channels http://example.com/user@example.com

Required nodes:

Topic channel

- posts

Personal channel

- posts
- geo/current
- geo/previous
- geo/future
- mood
- subsciptions

buddycloud Server Discovery

Server discovery happens per domain (the part of a JID that comes after the @ and before the /). Once a client has found its home buddycloud server, additional protocols are planned to leverage better connections and fuller caches of servers to find further remote buddycloud servers.

This is also used to check authority of a server for a user.

Determining compatibility

Channels capable components MUST advertise their type in order to be used by other entities:

```
<identity category="pubsub" type="channels"/>
```

Protocol flow

Using XEP-0030 Service Discovery:

- Fetch *items* for the domain
- Fetch *info* for each item
- Pick the service with the following identity. Note that a response can contain multiple <*identity*/> elements.

client discovers the component

```
<identity category='pubsub' type='channels'/>
```

the server replies with the component address

```
... something here
```

The buddycloud Inbox

The inbox is an auxiliary service that can be discovered similar to the content-hosting service by *pubsub/inbox*. It takes care of receiving all notifications and synchronization while providing a one-shot MAM protocol for replaying them to clients. Therefore, users need only one presence authorization to that inbox.

Inbox Discovery

Inbox components MUST advertise their type in order to be used by users:

```
<identity category="pubsub" type="inbox"/>
```

Authorization

Any stanza except subscribe/unsubscribe do not convey identity of the end-user.

The remote server has to check authority of the inbox/local server for the domain/user. It also has to check that at least one user of the inbox has permission to access the node.

The inbox/local server has to synchronize subscriptions/affiliations to decide permissioning for its local users.

Partition Detection

Channel services MUST also advertize all supported XMPP extensions in Service Discovery *feature* elements.

Network partition detection has been designed for the notification-sending side. A service does not need to track delivery status of each message, but keeps track of remote services that were unreachable.

When an inbox comes online after downtime, it has to retrieve all metadata and posts from user-subscribed nodes.

In case of a network partition, it is the notification sender that detects unavailability of an inbox. It keeps track of the services that were offline, and periodically sends:

```
<iq type="set" from="channels.denmark.lit" to="channels.shakespeare.lit">
  <you-missed-something xmlns='http://buddycloud.org/v1'/>
</iq>
```

Timestamp perservation

Server-to-server syncing should always preserve items, config, subscription, and affiliation timestamps. Otherwise a resync would cause MAM to send too much to clients.

Channel Management

Create a channel

If the user doesn't already have a channel, one can be created. The buddycloud server will then also setup the associated geoloc and mood nodes and ensure that the nodes have the correct permissions.

```
<iq from="channeluser@example.com/ChannelCompatibleClient" to="channelserver.example
  <query xmlns='jabber:iq:register'/>
```

```
</iq>
```

Support for registration can be determined by Service Discovery feature advertisements, ie:

```
<feature var='jabber:iq:register'/>
```

buddycloud to pubsub mappings

how does the spec map to the application

buddycloud	XEP-0060 mapping
channel producer	Owner
channel moderator	
channel follower+post	Publisher
channel follower	Member
web session via BOSH	Member / None (?)
banned follower	Outcast

Edit Channel Metadata

Each channel must have the following metadata set

Setting	Field	Example	Notes	Defaults
Channel title	pubsub#title	My buddycloud channel	displayed on a single line in clients	user@example.org's buddycloud channel
Channel description	pubsub#description	This is a channel about the things that I like to do.	a multi-line description in clients	<empty>
Access model	pubsub#access_model	whitelist	Who can view the channel? ▪ *whitelist* means the channel is only viewable by followers, followers+post and moderators. ▪ *open* means anyone can view the channel ▪ *open* makes the channel web-viewable to anyone	Channel posts are open to the world, location is open to just friends. i.e. ▪ .../posts - open ▪ .../mood - open ▪ .../geo/future - whitelist ▪ .../geo/current - whitelist ▪ .../geo/previous -whitelist

Default affiliation	buddycloud#default_affiliation	publisher	What role do new followers have? Spam problems in open topic channels mean that it's a good idea to only have new followrs unable to post until the channel producer understands the implications of anyone being able to post to their new channel or the new channel has enough moderators to remove spam posts. ▪ *publisher* (userspeak: "follower+post") ▪ *member* (userspeak:"follower")	▪ topic channels: follower ▪ personal channels: follower+post
Channel type	buddycloud#channel_type	personal	What type of channel is this? ▪ *topic:* channels not associated with a topic. ▪ *personal:* channels about a jid	Create channel issued by a jid will first create a channel for that jid.
Channel creation date	pubsub#creation_date	2011-01-31T23:59:42	When the channel was created in ISO 8601 time format	Sure an individual domain could spoof this. But that is sad. And pathetic.

Upon update buddycloud server SHOULD send configuration notifications.

Synchronization

Via Message Archive Management:

```
<iq type='get' id='sync1'
    from='francisco@denmark.lit/barracks'
    to='channels.denmark.lit'>
  <query xmlns='urn:xmpp:archive#management'
         start='2010-06-07T00:00:00Z'
         end='2002-07-07T13:23:54Z'/>
</iq>
```

The server now replays all notifications of the requested timespan, concluded by:

```
<iq type='result' id='sync1'
    to='francisco@denmark.lit/barracks'
    from='channels.denmark.lit'/>
```

In those notifications the server must send even unsubscribed subscription states for full synchronization. It must also send subscription updates prior post & configuration state so that clients are aware of what to track (the user's subscriptions).

Affiliation changes

When a client receives notification that its JID has an updated affiliation on a node, it should go out and refresh all data. This is because different roles see different users as described in Channel protocol. **FIXME:** this must also be done by inboxes!

Subscribing to a channel

client to inbox

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='channels.denmark.lit'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe node='/users/henry@shakespeare.lit/posts'
               jid='francisco@denmark.lit'/>
  </pubsub>
</iq>
```

inbox to federated service

```
<iq type='set'
    from='channels.denmark.lit'
    to='channels.shakespeare.lit'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe node='/users/henry@shakespeare.lit/posts'
               jid='francisco@denmark.lit'/>
    <actor xmlns='http://buddycloud.org/v1'>francisco@denmark.lit</actor>
  </pubsub>
</iq>
```

The remote service checks the actor for permission.

The *@jid* attribute is purely decorative to comply with XEP-0060. It is not the actual notification listener (in that case the inbox instead of the user himself), but the user that wishes to subscribe. Use of the *<actor/>* element is more generic and applies to a variety of other requests that require authorization.

Upon successful subscription/unsubscription, the inbox must keep track if there's at least one subscription to the node with the inbox itself as listener, allowing it to answer read queries from only its cache.

Inbox then receives acknowledgement for subscription request, and relays it to the client:

```
<iq type='result'
    from='channels.denmark.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
```

```
        node='/users/henry@shakespeare.lit/posts'
        jid='francisco@denmark.lit'
        subscription='subscribed'/>
  </pubsub>
</iq>
```

Subscribed inboxes and clients are also notified:

```
<message
    from='channels.shakespeare.lit'
    to='horatio@denmark.lit'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <subscription node='/users/henry@shakespeare.lit' jid='francisco@denmark.lit' su
  </event>
</message>
```

Temporary subscriptions

XEP-0060 makes it possible to have temporary subscriptions.

Servers MUST deny temporary subscriptions to private channels (with a *<not-allowed/>* stanza). They SHOULD send unsubscription requests to federated services when the temporary subscribed becomes unavailable. A server MAY delete temporary subscriptions at any time: when restarting, when a channel becomes private, after a certain amount of time, etc. So clients SHOULD NOT consider temporary subscriptions to be ultimately reliable.

When a client wants to subscribe temporarily to a channel, it MUST subscribe and configure.

Client to inbox:

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='channels.denmark.lit'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe node='/users/henry@shakespeare.lit/posts'
               jid='francisco@denmark.lit'/>
    <options node='/users/henry@shakespeare.lit/posts'
             jid='francisco@denmark.lit'>
      <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>
```

Inbox to federated service:

```
<iq type='set'
    from='channels.denmark.lit'
    to='channels.shakespeare.lit'
```

```
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe node='/users/henry@shakespeare.lit/posts'
               jid='francisco@denmark.lit'/>
    <actor xmlns='http://buddycloud.org/v1'>francisco@denmark.lit</actor>
    <options node='/users/henry@shakespeare.lit/posts'
             jid='francisco@denmark.lit'>
      <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>
```

If successful, the federated services sends an acknowledgement to the inbox, which relays it to the client. The <*subscription*/> element of this acknowledgement MUST include a *temporary* attribute:

```
<iq type='result'
    from='channels.denmark.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='/users/henry@shakespeare.lit/posts'
        jid='francisco@denmark.lit'
        subscription='subscribed'
        temporary='1'/>
  </pubsub>
</iq>
```

Servers MAY also include the *temporary* attribute for non-temporary subscription acknowledgements.

Servers MUST NOT send notifications about temporary subscriptions to federated services nor to the client, neither when subscribing nor when unsubscribing, as it wouldn't respect the privacy of people just visiting a channel without subscribing to it.

Toggle channel privacy

Set the channel access model. Who can view the channel?

- whitelist means the channel is only viewable by followers, followers+post and moderators.
- open means anyone can view the channel
- open makes the channel web-viewable to anyone

```
<iq type='set'
    from='channeluser@example.com/buddycloud-client'
    to='channelserver.example.com'
    id='create1'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
       <create node='/user/channeluser@example.com'/>
       <configure>
          <x xmlns='jabber:x:data' type='submit'>
             <field var='FORM_TYPE' type='hidden'>
                <value>http://jabber.org/protocol/pubsub#node_config</value>
             </field>
             <field var='pubsub#access_model'><value>whitelist</value></field>
```

```
            </x>
         </configure>
      </pubsub>
   </iq>
```

Alter the default affiliation

What affiliation do new channel followers have?

- publishers (userspeak: "follower+post")
- subscribers (userspeak: "follower")

```
<iq type='set'
    from='channeluser@example.com/ChannelCompatibleClient'
    to='channelserver.example.com'
    id='create1'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
       <create node='/user/channeluser@example.com'/>
       <configure>
          <x xmlns='jabber:x:data' type='submit'>
             <field var='FORM_TYPE' type='hidden'>
                <value>http://jabber.org/protocol/pubsub#node_config</value>
             </field>
             <field var='pubsub#publish_model'><value>publishers</value></field>
          </x>
       </configure>
    </pubsub>
</iq>
```

Anonymous channel access

Anonymous channel access is used by non-authenticated clients such as webclients or web widgets.

A service should discover an acting entity and deny any modifying requests if the entity has an identity of *account/anonymous* according to XEP-0175 Discovery.

Anonymous users may do temporary subscriptions to open channels. These subscriptions must be removed by the servers once the last resource of the anonymous user has sent unavailable presence.

Post management

Content normalization

When posting to a channel a server MUST apply the following rules to the ATOM payload:

- The `entry/author/uri` MUST be set to the **acct:** account of the entity that is actually posting to the channel. This shoudl be the bare jid.
- Upon a new entry the `entry/published` timestamp MUST be set to the current server time (often more accurate than client clock)
- Upon overwriting an existing entry, the `entry/published` SHOULD be copied from the old version and `entry/updated` SHOULD be set to the current server time
- The `entry/id` element SHOULD be updated to reflect the PubSub content model
- Any conversation thread information MAY be checked for existing references, and possibly deny posting with missing context
- The `entry/content` MAY be used for spam filtering, especially for posts from non-channel-owners

- Missing Activity Streams constructs may be replaced by verb **post** or **comment** (depending on presence of **thr:in-reply-to**)

Clients (especially in Web browsers) SHOULD NOT display rich content of `entry/content[@type='html' or @type='xhtml']` without sanitizing markup and scrubbing JavaScript

- If a server provides channel data via HTTP, it MAY add `entry/link[@rel='alternate']` pointers to these representations

Note: Add something about post size restrictions

Note: in a decentralized system not only content-hosting servers may validate content.

Create a post

Posting to a channel implies a role of "follower+post, moderator or producer"

```
<iq from="channeluser@example.com/ChannelCompatibleClient" to="channelserver.example
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
       <publish node="/user/koski@buddycloud.com/posts">
          <item>
             <entry xmlns="http://www.w3.org/2005/Atom" xmlns:activity="http://activ
                <published>2010-01-06T21:41:32Z</published>
                <author>
                   <name>koski@buddycloud.com</name>
                   <jid xmlns="http://buddycloud.com/atom-elements-0">koski@buddyclo
                </author>
                <content type="text">Test</content>
                <geoloc xmlns="http://jabber.org/protocol/geoloc">
                   <text>Paris, France</text>
                   <locality>Paris</locality>
                   <country>France</country>
                </geoloc>

                <activity:verb>post</activity:verb>
                <activity:object>
                  <activity:object-type>note</activity:object-type>
                </activity:object>
             </entry>
          </item>
       </publish>
    </pubsub>
 </iq>
```

The *activity:object-type* should be **comment** if the entry is a reply to another post.

The inbox server will relay the stanza to the hosting component along with **<actor/>** information.

Receive a post

Subscribed inboxes are always notified. Online clients automatically receive new posts from the inbox

```
<message type="headline" id="bc:GfLwH" from="channelserver.example.com" to="3194151
  <event xmlns="http://jabber.org/protocol/pubsub#event">
    <items node="/user/channeluser@example.com/posts">
      <item id="1291048810046">
        <entry xmlns="http://www.w3.org/2005/Atom" xmlns:thr="http://purl.org/syndic
          <author>
```

```
            <name>Dirk</name>
            <jid xmlns="http://buddycloud.com/atom-elements-0">fahrertuer@example.co
            <affiliation xmlns="http://buddycloud.com/atom-elements-0">moderator</af
          </author>
          <content type="text">A comment, wondering what all this testing does</cont
          <published>2010-11-29T16:40:10Z</published>
          <updated>2010-11-29T16:40:10Z</updated>
          <id>/user/channeluser@example.com/posts:1291048810046</id>
          <geoloc xmlns="http://jabber.org/protocol/geoloc">
            <text>Bremen, Germany</text>
            <locality>Bremen</locality>
            <country>Germany</country>
          </geoloc>
          <thr:in-reply-to ref="1291048772456"/>
        </entry>
      </item>
    </items>
  </event>
</message>
```

Retrieve a post

Node items are ordered newest first. The Result Set Management elements `after` and `before` are not related to timestamps but this ordering!

Client sends:

```
<iq type='get'
    from='channeluser@example.com/KillerApp'
    to='channelserver.example.com'
    id='items1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='/user/kitteh@example.com/posts'/>
  </pubsub>
</iq>
```

Server responds:

```
<iq type='result'
    from='channelserver.example.com'
    to='channeluser@example.com/KillerApp'
    id='items1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='/user/kitteh@example.com/posts'>
      <item id="1291048810046">
        <entry xmlns="http://www.w3.org/2005/Atom" xmlns:thr="http://purl.org/syndic
          <author>
            <name>Dirk</name>
            <jid xmlns="http://buddycloud.com/atom-elements-0">fahrertuer@example.co
            <affiliation xmlns="http://buddycloud.com/atom-elements-0">moderator</af
          </author>
          <content type="text">A comment, wondering what all this testing does</cont
          <published>2010-11-29T16:40:10Z</published>
          <updated>2010-11-29T16:40:10Z</updated>
          <id>/user/channeluser@example.com/posts:1291048810046</id>
          <geoloc xmlns="http://jabber.org/protocol/geoloc">
            <text>Bremen, Germany</text>
            <locality>Bremen</locality>
            <country>Germany</country>
          </geoloc>
```

```
          <thr:in-reply-to ref="1291048772456"/>
        </entry>
      </item>
      <!-- [more items...] -->
      <item id="1131048810046">
        <entry xmlns="http://www.w3.org/2005/Atom" xmlns:thr="http://purl.org/syndic
          <author>
            <name>Kittycat</name>
            <jid xmlns="http://buddycloud.com/atom-elements-0">kitteh@example.com</
            <affiliation xmlns="http://buddycloud.com/atom-elements-0">owner</affil
          </author>
          <content type="text">I'm in ur channel, meowing federatedly</content>
          <published>2009-08-12T16:40:10Z</published>
          <id>/user/channeluser@example.com/posts:1131048810046</id>
          <geoloc xmlns="http://jabber.org/protocol/geoloc">
            <text>Home basket</text>
            <locality>Munich</locality>
            <country>Germany</country>
          </geoloc>
        </entry>
      </item>
    </items>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first>1291048810046</first>
      <last>1131048810046</last>
      <count>19827</count>
    </set>
  </pubsub>
</iq>
```

Client can then request older items for pagination or filling its cache:

```
<iq type='get'
    from='channeluser@example.com/KillerApp'
    to='channelserver.example.com'
    id='items2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='/user/kitteh@example.com/posts'/>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>1200</max>
      <after>1131048810046</after>
    </set>
  </pubsub>
</iq>
```

Result Set Management is strongly needed for chunkifying information. Even if a user is requesting 10000 items, a buddycloud server should never send stanzas exceeding 64KB, or else XMPP servers will kill the (component or s2s) connection.

Retrieve recent posts from all subscribed channels

Clients that need to access recent posts in all the followed channels may use the quick synchronization mechanism. This can only be used by authenticated users.

Client sends:

```
<iq from='channeluser@example.com/ChannelCompatibleClient' to='channelserver.example
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <recent-items xmlns='http://buddycloud.org/v1'
```

```
                since='2012-12-04T23:36:51.123Z'
                max='50'/>
  </pubsub>
</iq>
```

The servers responds with items from the subscribed channels:

- items from the `/posts` nodes only
- updated after `since`, *newest items first*, **required**
- at most `max` per channel, **required**
- RSM must be used to page through results

```
<iq type='result'
    from='channelserver.example.com'
    to='channeluser@example.com/ChannelCompatibleClient'
    id='ri1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='/user/kitteh@example.com/posts'>
      <item id="1291048810046">
        <entry>...</entry>
      </item>
      <!-- [more items...] -->
    </items>
    <items node='/user/channeluser@example.com/posts'>
      <!-- [some items]-->
    </items>
    <items node='/user/kitteh@example.com/posts'>
      <!-- [older items from that node] -->
    </items>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first>/user/kitteh@example.com/posts;1291048810046</first>
      <last>/user/tut@example.com/posts;1131048810046</last>
      <count>19827</count>
    </set>
  </pubsub>
</iq>
```

Retrieve all replies from a single post

This stanza retrieve all replies from a single post. It is useful in cases the user has new replies in old posts, so that he does not need to look down the node stream to fetch old replies. This can only be used by authenticated users. The attrs node and item_id are mandatory.

Client sends:

```
<iq type="get" from="channeluser@example.com/ChannelCompatibleClient'" to="channelse
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <replies xmlns="http://buddycloud.org/v1" node="/user/channeluser@example.com/po
  </pubsub>
</iq>
```

The servers responds with replies to that item

- RSM must be used to page through results

```
<iq type='result'
    from='channelserver.example.com'
    to='channeluser@example.com/ChannelCompatibleClient'
    id='ri1'>
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <items>
      <item id="1291048810046">
        <entry>...</entry>
      </item>
      <item id="1287390194809">
        <entry>...</entry>
      </item>
      ...
    </items>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first>/user/kitteh@example.com/posts;1291048810046</first>
      <last>/user/tut@example.com/posts;1131048810046</last>
      <count>19827</count>
    </set>
  </pubsub>
</iq>
```

Delete a post

The client sends

```
<iq from="channeluser@example.com/ChannelCompatibleClient" to="channelserver.example
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <retract node="/user/channeluser@example.com/posts" notify="1">
      <item id="1291048772456"/>
    </retract>
  </pubsub>
</iq>
```

Server replies

```
<iq from="channeluser@example.com/ChannelCompatibleClient" to="channelserver.example
```

A retraction message is sent to all online clients, along with an Atom tombstone to replace the deleted post

```
<message from="channelserver.example.com" id="bc:MGV3B" to="imranraza@example.com">
  <event xmlns="http://jabber.org/protocol/pubsub#event">
    <items node="/user/channeluser@example.com/posts">
      <retract id="1291048772456"/>
      <item id="1291048772456">
        <deleted-entry xmlns="http://purl.org/atompub/tombstones/1.0" ref="xmpp:char
          <updated>2012-07-01T15:08:32.950Z</updated>
          <id xmlns="http://www.w3.org/2005/Atom">1291048772456</id>
          <link xmlns="http://www.w3.org/2005/Atom" href="xmpp:channels.example.com?
          <published xmlns="http://www.w3.org/2005/Atom">2012-07-01T15:08:30.922Z</p
          <object xmlns="http://activitystrea.ms/spec/1.0/">
            <object-type>comment</object-type>
          </object>
          <verb xmlns="http://activitystrea.ms/spec/1.0/">post</verb>
        </deleted-entry>
      </item>
```

```
    </items>
  </event>
</message>
```

buddycloud Firehose

The **/firehose** node is a realtime relay feed of all posts from open channels and can be used with feed aggregation services like Superfeedr.

Follower Management

Follower Roles

The following roles exist in buddycloud

Users see	Reported as	Capabilities	Notes
producer	owner	publish and delete anyone's posts	
moderator	moderator	posts can remove posts and approve new followers	
follower+post	publisher	read and post	default role for new followers
follower	member	read-only	
banned	outcast	no posting rights. new follow requests from this user are also ignored	To avoid "glorification of the bad", only the channel producer and moderators should see the banned list.

Actions allowed for each role: https://buddycloud.org/wiki/buddycloud_Application_Logic#Visibility

Servers should enforce the following rules for channel followers:

- Each channel follower should be just listed once (a user cannot be a channel owner and banned for example)
- Each channel can have just one owner
- If the channel is a personal channel then that owner must match the channel JID

Query for information about a user

The client sends:

```
<iq from="channeluser@example.com/ChannelCompatibleClient" to="channelserver.example
  <query xmlns="http://jabber.org/protocol/disco#info" node="/user/channeluser@examp
</iq>
```

The server replies:

```
<iq type="result" id="metadata1" from="channelserver.example.com" to="channeluser@ex
  <query xmlns="http://jabber.org/protocol/disco#info" node="/user/channeluser@examp
```

```
    <identity category="pubsub" type="channel"/>
    <feature var="http://jabber.org/protocol/pubsub"/>
    <x xmlns="jabber:x:data" type="result">
      <field var="FORM_TYPE" type="hidden">
        <value>http://jabber.org/protocol/pubsub#meta-data</value>
      </field>
      <field var="buddycloud#subscribers-count" label="Number of subscribers" type='
        <value>23</value>
      </field>
      <field var="buddycloud#subscriptions-count" label="Number of channels the use
        <value>42</value>
      </field>
      <field var="buddycloud#moderates-count" label="Number of channels the user mo
        <value>8</value>
      </field>
      <field var="buddycloud#produces-count" label="Number of channels the user pos
        <value>13</value>
      </field>
    </x>
  </query>
</iq>
```

Retrieve followers

Query for the followers of a channel.

The client sends:

```
<iq from="channeluser@example.com/ChannelCompatibleClient" to="channelserver.example
  <pubsub xmlns="http://jabber.org/protocol/pubsub#owner">
    <affiliations node="/user/channeluser@example.com/posts"/>
    <set xmlns="http://jabber.org/protocol/rsm">
      <max>30</max>
    </set>
  </pubsub>
</iq>
```

The server replies with a list of users and their affiliations:

```
<iq type="result" id="26:85" from="channelserver.example.com" to="channeluser@examp
  <pubsub xmlns="http://jabber.org/protocol/pubsub#owner">
    <affiliations node="/user/channeluser@example.com/posts">
      <affiliation jid="175341640128870418445981 9@anon.buddycloud.com" affiliation='
      <!-- anon users are usually bosh based read-only web views -->
      <affiliation jid="akioh@anotherdomain.com" affiliation="publisher"/>
    </affiliations>
  </pubsub>
</iq>
```

- Change the affiliation of a channel follower

Following

In order to make a user's subscription & affiliation state on remote servers discoverable, a node on the home server needs to be filled. Use item ids by remote user to selectively update.

```
<iq from="channeluser@example.com/ChannelCompatibleClient" to="channelserver.example
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
     <publish node="/user/channeluser@example.com/subscriptions">
        <item id="koski@buddycloud.com">
          <query xmlns="http://jabber.org/protocol/disco#items" xmlns:pubsub="http:/
            <item jid="sandbox.buddycloud.com"
                  node="/user/koski@buddycloud.com/posts"
                  pubsub:affiliation="publisher">
              <atom:updated>2010-12-26T17:30:00Z</atom:updated>
            </item>
            <item jid="sandbox.buddycloud.com"
                  node="/user/koski@buddycloud.com/geo/future"/>
            <item jid="sandbox.buddycloud.com"
                  node="/user/koski@buddycloud.com/geo/current"/>
            <item jid="sandbox.buddycloud.com"
                  node="/user/koski@buddycloud.com/geo/previous"/>
            <item jid="sandbox.buddycloud.com"
                  node="/user/koski@buddycloud.com/mood"
                  pubsub:affiliation="member"/>
          </query>
        </item>
     </publish>
  </pubsub>
</iq>
```

Clients may also use this information to accelerate discovery of remote servers/subscriptions. Actual states as defined by the hosting (remote) server should be synchronized nevertheless.

The **atom:updated** element does not serve synchronization (which is based on client-side cache state). It indicates the timestamp of the last post a user read **for showing number of unread items**.

Follower Authorization

Subscription changes to a state of *pending* are **not** included in standard subscription change notifications.

Under an access model of **authorize**, the PubSub mechanisms are used. Contrary to iq requests, *actor* information is put in a form field:

```
<message to='inbox.denmark.lit' from='channels.denmark.lit'>
  <x xmlns='jabber:x:data' type='form'>
    <title>PubSub subscriber request</title>
    <field var='FORM_TYPE' type='hidden'>
      <value>http://jabber.org/protocol/pubsub#subscribe_authorization</value>
    </field>
    <field var='pubsub#node' type='text-single' label='Node ID'>
      <value>/user/hamlet@denmark.lit/posts</value>
    </field>
    <field var='pubsub#subscriber_jid' type='jid-single' label='Subscriber Address'>
      <value>spy@nsa.gov</value>
    </field>
    <field var='pubsub#allow' type='boolean'
           label='Allow spy@nsa.gov to subscribe to the posts of hamlet@denmark.lit?
      <value>false</value>
    </field>
  </x>
</message>
```

```
<message from='inbox.denmark.lit' to='channels.denmark.lit' id='approve1'>
  <x xmlns='jabber:x:data' type='submit'>
    <field var='FORM_TYPE' type='hidden'>
      <value>http://jabber.org/protocol/pubsub#subscribe_authorization</value>
    </field>
    <field var='pubsub#node'>
      <value>/user/hamlet@denmark.lit/posts</value>
    </field>
    <field var='pubsub#subscriber_jid'>
      <value>spy@nsa.gov</value>
    </field>
    <field var='pubsub#allow'>
       <value>false</value>
    </field>
    <field var='buddycloud#actor'>
       <value>hamlet@denmark.lit</value>
    </field>
  </x>
</message>
```

There's no real use in the process pending requests. Instead, use the notifications above. Include them in MAM replays too.

- This page was last modified on 4 October 2013, at 14:45.