XEP-0369: Mediated Information eXchange (MIX)

Kevin Smith
mailto:kevin.smith@isode.com
xmpp:kevin.smith@isode.com

Steve Kille
mailto:steve.kille@isode.com
xmpp:steve.kille@isode.com

Peter Saint-Andre
mailto:xsf@stpeter.im
xmpp:peter@jabber.org
http://stpeter.im/

2018-01-08
Version 0.9.5

Status	Type	Short Name
Experimental	Standards Track	MIX

This document defines Mediated Information eXchange (MIX), an XMPP protocol extension for the exchange of information among multiple users through a mediating service. The protocol can be used to provide human group communication and communication between non-human entities using channels, although with greater flexibility and extensibility than existing groupchat technologies such as Multi-User Chat (MUC). MIX uses Publish-Subscribe to provide flexible access and publication, and uses Message Archive Management (MAM) to provide storage and archiving.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

The Mediated Information eXchange (MIX) protocol is intended as a replacement for Multi-User Chat (MUC). MUC is a major application of XMPP that was developed in 2002 and standardized in Multi-User Chat (XEP-0045) [1]. MIX implements the same basic MUC patterns in a more flexible and extensible way in order to address requirements that have emerged since MUC was developed. MIX supports all of the core chatroom features that are familiar from MUC, such as discussion topics and invitations. Like MUC, it also defines a strong access control model, including the ability to ban users, to name administrators, and to provide controls as to which users can participate in channels.
Several reasons exist for replacing MUC:

- A number of use cases for group communication have emerged since MUC was first published.
- Experience has shown that it is difficult to use MUC to build several kinds of communication applications (such as a multimedia conference) without undesirable adaptations.
- It is impractical to address a number of the requirements listed in the next section with MUC or with extensions to MUC.
- In the years after MUC was designed, both Publish-Subscribe (XEP-0060) [2] and Message Archive Management (XEP-0313) [3] have been developed and it is desirable to reuse these building blocks (e.g., MAM can be used for message history) rather than using the less robust methods defined in Multi-User Chat (XEP-0045) [4].

Because it is anticipated that there will be significant co-existence between MUC and MIX, this specification is designed so that:

- XMPP clients can implement MUC and this specification in a way that provides a coherent user experience.
- XMPP servers can implement this specification and also provide a MUC interface in order to support clients that only implement MUC.

This specification gives guidance on supporting both MUC and MIX representations of chatrooms.

[1] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[2] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[3] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
[4] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

2 Requirements

The following requirements have been identified, which MIX aims to address.

1. A user's participation in a channel persists and is not modified by the user's client going online and offline.
2. Multiple devices associated with the same account can share the same nick in the channel, with well-defined rules making each client individually addressable.
3. Channels are NOT REQUIRED to support or reflect presence for participants.
4. A reconnecting client can quickly resync with respect to messages and presence.
5. A user MAY (subject to configuration) receive messages from a channel as an invisible observer.
6. Configuration can be observed externally to the channel (e.g., list of participants, access control rights, etc.).
7. MIX services SHOULD provide mechanisms to prevent JIDs from being harvested.
8. MIX and Message Archive Management (MAM) MUST work well together.
9. A user can determine which channels they participate in.
10. Provide extensibility regarding data formats that can be sent within a channel (files, structured data, indications about media sources in multimedia conferences, etc.) as well as flexibility regarding which data formats a user wants to receive.
11. Enable federation of a channel across multiple servers, to provide a service equivalent to "federated MUC" Federated MUC for Constrained Environments (XEP-0289) [5].
12. Enable sharing of messages on a channel without requiring sharing of presence.
13. Enable sharing of presence without requiring message sending.
14. (Desirable) Make it easier to reduce duplicate traffic.

3 Concepts

3.1 Specification Appproach

MIX will enable a wide range of auxiliary services. The goal of the MIX specification is to set out the core capabilities needed for MIX. It is anticipated that additional XEPs will be written to extend the core MIX, and the core MIX specification notes some areas where this may

[5] XEP-0289: Federated MUC for Constrained Environments <https://xmpp.org/extensions/xep-0289.html>.

happen. This approach will avoid the core MIX specification becoming unduly large. Profiles referencing sets of related MIX XEPs may be developed in the future.

3.2 Core Concepts

The following concepts underlie the design of MIX.

1. MIX channels (roughly equivalent to MUC rooms) are hosted on one or more MIX domains, (examples: 'mix.example.com'; 'conference.example.com'; 'talk.example.com'), which are discoverable through Service Discovery (XEP-0030) [6]. Each channel on a MIX service can then be discovered and queried.

2. In MIX each channel (e.g., 'channel@mix.example.com') is a specialized pubsub service. This is based on the model from Personal Eventing Protocol (XEP-0163) [7] where every user JID (e.g., 'user@example.com') is its own pubsub service.

3. A channel's pubsub service contains a number of nodes for different event types or data formats. As described below, this document defines several standard nodes; however, future specifications or proprietary services can define their own nodes for extensibility.

4. Affiliations with the nodes are managed by the MIX service by channel level operations, so that the user does not have to separately manage affiliations with the individual PubSub nodes.

5. Message Archive Management (XEP-0313) [8] (MAM) is used for all history access, with each node being individually addressable for MAM queries. This simplifies implementation compared to MUC (which had a specialized and rather limited history retrieval mechanism).

6. A client can achieve a 'quick resync' of a node by requesting just those changes it has not yet received, using standard MAM protocol. This solves the MUC issue of either receiving duplicate messages when rejoining a room or potentially missing messages.

7. Because MAM is used for history, only those nodes that have a 'current value' need to store any items in them — e.g., 'urn:xmpp:mix:nodes:presence' and 'urn:xmpp:mix:nodes:information' would store their current values (with older values being queryable through MAM), while 'urn:xmpp:mix:nodes:messages' would store no items.

8. A user's participation in a channel outlives their client sessions. A client which is offline will not share presence within the channel, but the associated user will still be listed as an participant.

[6] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[7] XEP-0163: Personal Eventing Protocol <https://xmpp.org/extensions/xep-0163.html>.
[8] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.

9. Presence is sent to participants using bare JID, whether or not the user has an online client.

10. Online clients MAY register presence, which is then shared with participants who have subscribed to presence.

11. MIX decouples addressing of channel participants from their nicknames, so that nickname changes do not affect addressing.

12. Each participant is addressable by a single bare JID, which is a proxy JID (not the user's real JID) to make it straightforward to hide the user's real JID from other channel participants. Full JIDs comprised of this bare JID plus a resource (also anonymized) are then constructed, allowing visibility into the number of online resources participating in a channel.

13. Although some protocol is shared with MUC, MUC clients are not interoperable with a MIX service.

14. MIX requires the server to which the MIX client connects to provide functionality to support MIX. This functionality is defined in this specification and referenced as "MIX Participant's Server Behaviour".

15. MIX domains MUST NOT be used to host end user JIDs.

3.3 MIX and PubSub

MIX is based upon domains providing a MIX service, such as 'mix.shakespeare.example'. Note that although PubSub communication is used, a domain used for MIX is a MIX domain and not a standard Publish-Subscribe (XEP-0060) [9] domain. Like MUC, there is no requirement on the naming of these domains; the label 'mix' used in examples in this specification and the fact that it is a subdomain of a 'shakespeare.example' service are purely for example.
Every MIX channel is an addressable service based on PubSub (with additional MIX semantics) that will be addressed using a bare JID by other XMPP entities, for example coven@mix.shakespeare.example. While Publish-Subscribe (XEP-0060) [10] is used as the basis for the MIX model, MIX uses standard presence and groupchat messages to provide an interface to the MIX service that does not expose PubSub protocol for many of the more common functions.

3.4 MIX and MAM

Historical data (such as the messages sent to the channel) is stored in an archive supporting Message Archive Management (MAM) so that clients can subsequently access this data with MAM. Each node can be archived separately (e.g., the presence node or the configuration

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

node). MIX messages are archived by both the MIX channel and the user's server, so that clients can generally use their local MAM archiver. MIX clients can retrieve archived information with MAM in order to quickly resync messages with regard to a channel, and can do so without providing presence information.

3.5 Behaviour of MIX Participant's Server

A MIX channel does not send messages and presence directly to the MIX client of a channel participant, but addresses them to the participant using the participant's bare JID. The participant's server MUST then handle these messages and pass them on to zero, one or multiple clients. To enable MIX to work, this behaviour is necessary and so the server of every MIX client MUST follow the rules set out in this specification. This approach enables flexible support of multiple clients for a MIX channel participant. The MIX model is that a user will join a channel over an extended period, and that the user (not a specific client used by the user) joins the channel. The primary subscription is with the client's bare JID. There are a number of MIX requirements on behaviour of the MIX Participant's server, which are summarized here:

1. Messages from a MIX client to a MIX channel will go direct to the MIX service. They are relayed, but not modified, by the participant's server.
2. Messages from a MIX channel to a MIX client are always sent to the MIX participant's server (addressed by bare JID) and MUST be handled by the MIX participant's server.
3. All MIX messages received by the MIX participant's server for a participant MUST be stored using MAM in the participant's archive.
4. The MIX participant's server will only forward messages to online clients and will not forward messages if no clients are online. This means that a MIX client needs to resynchronize with all MIX channels when it comes online. This message synchronization will happen between the MIX client and the MAM archive held on the MIX participant's server.
5. The MIX client will generally send management and other messages directly to the MIX channel and this MUST be done except where specifically noted.
6. The user's roster contains each MIX channel to which the user is subscribed. To achieve this the user's server needs to manage the roster on behalf of the user. For this reason, MIX join and leave commands MUST be sent by a client to the user's server. This enables the user's server to correctly manage the roster on behalf of the user.

Messages from a MIX channel to a MIX participant (which will be of type=groupchat) and presence information are sent to the participant's server using the participant's bare JID. This means that the MIX participant's server MUST implement a modification of the standard RFC

6121 [11] message processing rules.
The core MIX specification sets out how the MIX channel interacts directly with XMPP clients and with the MIX participant's server. Behaviour of the MIX participant's server is also specified in this MIX specification.

3.6 User Presence in MIX

MIX channels store presence of each online client for a user that chooses to publish presence. Presence is stored in the presence node and is encoded using a full proxy JID. Where a user publishes presence for one or more clients, this information is available to all users subscribing to the channel presence.
Channel participants can send messages to clients publishing their presence by sending them to the full proxy JID published in the presence node. These stanzas MAY be routed to the client by the MIX channel. The choice to do this is dependent on MIX channel policy. For example, a disco request MAY be routed through the MIX channel to the client.
Presence updates are distributed by a channel to the bare JID of subscribers and then the subscriber's server will distribute to each of the subscriber's currently online clients.

3.7 Private Messages

Private messages MAY be sent to channel participants if allowed by channel policy. Private messages are addressed to the user's bare proxy JID determined from the participants node, and routed by the MIX to the user's bare real JID, where standard distribution rules will apply.

3.8 Proxy JIDs and JID Visibility

MIX channels have three modes controlling JID visibility:

Mode	Description
'JID Visible'	In these channels all participant JIDs are visible to all channel participants.
'JID Maybe Visible'	In these channels, participant JIDs are visible to all channel participants when participant preference allows.
'JID Hidden'	In these channels, no participant JIDs are visible to channel participants, but they are visible to channel administrators.

[11] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

A channel participant MAY specify a preference for JID visibility for the channel, with one of the following values:

Preference	Description
'Prefer Visible'	The users JID will be visible if the channel is JID Visible or JID Maybe Visible channels and hidden if the channel is JID Hidden.
'Prefer Hidden'	The user's JID will be hidden if the channel is JID Maybe Visible and shown if the channel is JID Visible .
'Enforce Hidden'	The user's JID will never be shown and the user will be removed from channel if channel mode is changed to JID Visible.
'Enforce Visible'	The users JID will always be shown and the user will be removed from channel if mode is changed to 'JID Hidden'.

The default value is 'Prefer Hidden'.
The primary representation of a participant in a MIX channel is a proxy JID, which is an anonymized JID using the same domain as the channel and with the name of the channel encoded, using the format "<generated identifier>#<channel>@<mix domain>". The generated identifier MUST NOT contain the '#', '/' or '@' characters. The Channel name MAY contain the '#' character. For example in the channel 'coven@mix.shakespeare.example', the user 'hag66@shakespeare.example' might have a proxy JID of '123456#coven@mix.shakespeare.example'. The reason for the proxy JID is to support JID Hidden channels. Proxy JIDs are also used in JID Visible channels, for consistency and to enable switching of JID visibility. The "urn:xmpp:mix:nodes:jidmap" node maps from proxy JID to real JID. Servers and clients MUST determine that a JID is a proxy JID from context and MUST NOT infer that a JID is a proxy JID because it contains the '#' character.
The mapping of real bare JID to proxy JID is defined when a participant joins a channel. While a user is a participant in a Channel, the mapping between real JID and proxy JID MUST NOT be changed. This mapping must be maintained after the user leaves the channel. Proxy JID values MUST NOT be re-used. If a JID that left a channel joins the channel again the same proxy JID MAY be used again or a different new proxy JID MAY be assigned.
The example real full JIDs in this specification are aligned the anticipated new format that splits the resource into two parts. The first part is UUID that is a stable and fixed value for the client and is anticipated to be a fixed format which may well be UUID 4. The second part is server assigned and will vary with each session. A realistic JID with resource is: 'hag66@shakespeare.example/d104f6a7-97e9-477f-8947-e4a37691d7ee/7533375f2cd'. This specification uses examples of the style: 'hag66@shakespeare.example/UUID-a1j/7533' which are aligned to real resources, but more compact. CITATION TO BE ADDED. If the final specification differs from this, the examples will be updated.
The full JIDs held in the presence nodes are anonymized using a similar mechanism. First the bare JID is mapped to a bare proxy JID, using the mapping held in the "urn:xmpp:mix:nodes:jidmap" node for the bare JID. Then the resource is replaced with

a generated value. For example, 'hag66@shakespeare.example/UUID-a1j/7533' in the channel coven might have a proxy JID of '123456#coven@mix.shakespeare.example/789'. There is no client visible mapping of proxy full JIDs maintained as this is not needed. The MIX channel will need to maintain a mapping, to support directly addressing clients, such as for client to client disco#info queries. While an full proxy JID is held in the presence node, the mapping to real JID MUST NOT be changed. When adding a client to the presence node, the server MAY add the same anonymized JID as used before by that client, or it MAY create a different anonymized JID.

3.9 Standard Nodes

MIX defines a number standard nodes are as follows. Note that all nodes are OPTIONAL and that not every channel will necessarily use each node:

Name	Node	Description	Update	Distribution
Messages	'urn:xmpp:mix:nodes:messages'	Includes messages being sent to the channel. Each item of this node will contain a message sent to the channel.	Message	Message
Participants	'urn:xmpp:mix:nodes:participants'	Includes the list of participants and the associated nick. Channel participants are added when they join the channel and removed when they leave	Join/Leave/Set Nick	PubSub

Name	Node	Description	Update	Distribution
JID Map	'urn:xmpp:mix:nodes:jidmap'	Sharing a list of bare proxy JIDs from the participants node with a 1:1 mapping to the corresponding real JIDs.	Automatic	PubSub
JID Maybe Visible Map	'urn:xmpp:mix:nodes:jidmap-visible'	Sharing a list of bare proxy JIDs from the participants node with a 1:1 mapping to the corresponding real JIDs for participants that choose to share real JIDs in a channel with JID Maybe Visible mode.	Automatic	PubSub
Presence	'urn:xmpp:mix:nodes:presence'	Sharing information about the availability status of online participants, which MAY include multiple clients for a single participant.	Presence	Presence

Name	Node	Description	Update	Distribution
Information	'urn:xmpp:mix:nodes:info'	For storing general channel information, such as description.	PubSub	PubSub
Allowed	'urn:xmpp:mix:nodes:allowed'	For storing JIDs that are allowed to be channel participants.	PubSub	PubSub
Banned	'urn:xmpp:mix:nodes:banned'	For storing JIDs that are not allowed to be channel participants.	PubSub	PubSub
Configuration	'urn:xmpp:mix:nodes:config'	For storing channel configuration.	PubSub	PubSub

All of the standard nodes are OPTIONAL. A channel providing a service similar to MUC will typically use all of the standard nodes, but other channels MAY use any combination of these nodes. MIX provides mechanisms to allow users to conveniently subscribe to a chosen set of nodes and to unsubscribe to all nodes with a single operation. Some nodes are accessed and managed with PubSub, whereas other nodes define MIX specific mechanisms for their use, as shown in the last two columns of the table.
MIX also makes use of two nodes for support of Avatars. These nodes and their use is defined in User Avatar (XEP-0084) [12]. These nodes MAY be created as part of a MIX channel, where it is desired to publish an avatar associated with the channel.

Name	Node	Description
Avatar Data	'urn:xmpp:avatar:data'	For publishing an Avatar
Avatar Metadata	'urn:xmpp:avatar:metadata'	For publishing Avatar metadata

The structure of each of the standard nodes defined by MIX is now considered in more detail in the rest of this section, after explaining roles.

[12] XEP-0084: User Avatar <https://xmpp.org/extensions/xep-0084.html>.

3.9.1 Roles

There are a number of MIX roles for each channel, listed in the following table. Rights will be assigned to the various roles in the channel configuration node.

Role	Membership and Rights
Owners	These are owners of the channel, as specified in the channel configuration node. Only owners are allowed to modify the channel configuration node.
Administrators	Administrators are defined in the channel configuration node. Administrators have update rights to the Allowed Node and Banned Node, so they can control which users are allowed to participate in a channel.
Participants	Participants are users listed by JID in the participants node.
Allowed	Allowed is the set of JIDs that are participants or are allowed to become participants. A JID is allowed if it does not match an entry in the banned node and either it matches an entry in the allowed node or the allowed node is not present.
Anyone	Any user, including users in the banned node.

There MUST always be at least one Owner for a Channel. Owners, Administrators, Participants, and Allowed are optional and do not need to be set. Where no owner is explicitly set, it is anticipated that a server administrator will have owner rights. Rights are defined in a strictly hierarchical manner following the order of this table, so that for example Owners will always have rights that Administrators have.

3.9.2 Node Archiving

Nodes MAY be archived and where this is done MAM MUST be used. Archiving of the Messages node MUST be done as part of the MIX specification. Archiving of other nodes is OPTIONAL.

3.9.3 Messages Node

The Messages node is used to distribute messages. The Messages node is a transient node and so no PubSub items are held. Messages MUST go to the associated MAM archive and history is retrieved by use of MAM. Users subscribe to this node to indicate that messages from the channel are to be sent to them. Private Messages are not distributed by the Messages node.

3.9.4 Participants Node

Each channel participant is represented as an item of the 'urn:xmpp:mix:nodes:participants' channel node. Each item is named by the bare proxy JID of the participant. For example

'123456#coven@mix.shakespeare.example' might name the node item associated with participant 'hag66@shakespeare.example'. Information is stored in a <participant/> element qualified by the 'urn:xmpp:mix:1' namespace. The nick associated with the user is optional and is stored in a <nick/> child element of the <participant/> element. The nick for each channel participant MUST be different to the nick of other participants.
When a user joins a channel, the user's bare proxy JID is added to the participants node by the MIX service. When a user leaves a channel, the user's bare proxy JID is removed from the participants node. The participants node MUST NOT be directly modified using pubsub.
Users MAY subscribe to and read information this node, when permitted by the channel. Standard PubSub access will allow a full list of participants and associated nicks to be determined. By subscribing to the node, a user will be informed of changes to the Participants Node.
The participants node is OPTIONAL. If the Participants Node is not used, information on channel participants is not shared. If there is no participants node, the access control value 'participants' MUST NOT be used. The Participants node is a permanent node with one item per participant.

Listing 1: Value of Participants Node

```
<items node='urn:xmpp:mix:nodes:participants'>
  <item id='123456#coven@mix.shakespeare.example'>
      <participant xmlns='urn:xmpp:mix:1'>
         <nick>thirdwitch</nick>
      </participant>
  </item>
</items>
```

3.9.5 JID Map Node

The JID Map node is used to associate a proxy bare JID to its corresponding real bare JID. It is a PubSub node with the 'node' attribute set to 'urn:xmpp:mix:nodes:jidmap'. The JID Map node MUST have one entry for each entry in the Participants node. This value is added when a user joins the channel and is removed when the user leaves the channel. Each item is identified by proxy bare JID, mapping to the real bare JID. This node is used to give administrator access to real JIDs and participant access to real JIDs in jid-visible channels. This node MUST NOT be modified directly using pubsub. In JID Visible channels, all participants MAY subscribe to this node. In JID Hidden and JID Maybe Visible channels, only administrators can subscribe. The JID Map node is a permanent node with one item per participant. Information is stored in a <participant/> element qualified by the 'urn:xmpp:mix:1' namespace. The real JID is stored in a <jid/> child element of the <participant/> element.

Listing 2: Value of JID Map Node

```
<items node='urn:xmpp:mix:nodes:jidmap'>
  <item id='123456#coven@mix.shakespeare.example'>
```

```
      <participant xmlns='urn:xmpp:mix:1'>
         <jid>hecate@mix.shakespeare.example</jid>
      </participant>
  </item>
</items>
```

3.9.6 JID Maybe Visible Map Node

The JID Maybe Visible Map node is a similar node to the JID Map node that is used in addition to the JID Map Node in JID Maybe Visible channels. It is a PubSub node with the 'node' attribute set to 'urn:xmpp:mix:nodes:jidmap-visible'. All participants may subscribe to and access this node. It uses the same encoding as JID Map node and all participant JIDs MUST be included. Where a participant's preference is to not share the JID, the encoded participant value is the proxy JID. This will enable a user looking up a JID to clearly determine that the user preference is to not share the JID and to clearly distinguish this case from an erroneous proxy JID.

3.9.7 Presence Node

The presence node contains the presence value for clients belonging to participants that choose to publish presence to the channel. A MIX channel MAY require that all participants publish presence. Each item in the presence node is identified by the full proxy JID, and contains the current presence value for that JID. The presence is encoded in the same way as data that would be sent in a presence stanza using a <presence/> element qualified by the 'jabber:client' namespace. The full proxy JID is always used in this node. In MIX it is possible to have a 'presence-less channel' by not using this node. Access Control MAY be set to enforce that for each of the full JIDs in this list, the bare JID MUST be in the participants list.
This node MAY be subscribed to by users and this subscription MUST use the user's bare JID. So presence of online clients is sent to the user's server for each user subscribed to this node. Presence is always sent using standard presence protocol and NOT using pubsub protocol. Clients MUST NOT directly access the presence node using pubsub. The Presence node is a permanent PubSub node.

Listing 3: Value of Presence Node

```
<items node='urn:xmpp:mix:nodes:presence'>
  <item id='123456#coven@mix.shakespeare.example/8765'>
    <presence xmlns='jabber:client'>
      <show>dnd</show>
      <status>Making a Brew</status>
    </presence>
  </item>
</items>
```

3.9.8 Information Node

The Information node holds information about the channel. The information node contains a single item with the current information. The information node is named by the date/time at which the item was created. The information node is accessed and managed using standard pubsub. The Information node is a permanent node with a maximum of one item. Users MAY subscribe to this node to receive information updates. The Information node item MAY contain the following attributes, each of which is OPTIONAL:

Name	Description	Field Type	Values	Default
'Name'	A short string providing a name for the channel. For example "The Coven". Typical uses of this value will be to present a user with the name of this channel in a list of channels to select from or as the header of UI display of the channel. It is intended for use where a short string is needed to represent the channel.	text-single	-	-

Name	Description	Field Type	Values	Default
'Description'	A longer description of the channel. For example "The Place where the Macbeth Witches Meet up". A typical use would be to provide a user with more information about the channel, for example in a tool tip.	text-single	-	-
'Contact'	The JID or JIDs of the person or role responsible for the channel.	jid-multi	-	-
'JID Visibility'	Specified JID visibility of the channel.	list-single	'jid-hidden'; 'jid-maybe-visible'; 'jid-visible'	'jid-hidden'

Name and Description of the channel apply to the whole channel and will usually be changed infrequently.
JID visibility is included in the Information Node as this is information that will be useful for participant clients and may also be important when choosing to join a channel.
The name and description values MUST contain a "text" element and MAY contain additional text elements. Where multiple text elements are provided, each MUST posses an xml:lang attribute that describes the natural language of the element. The format of the Information node follows Data Forms (XEP-0004) [13]. This allows configuration to be updated by MIX defined commands and that the results of update commands are the same as the PubSub node. The following example shows the format of a item in the information node for the example coven@mix.shakespeare.example channel.

Listing 4: Information Node

```
<items node='urn:xmpp:mix:nodes:info'>
  <item id='2016-05-30T09:00:00'>
      <x xmlns='jabber:x:data' type='result'>
        <field var='FORM_TYPE' type='hidden'>
```

[13] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

```
            <value>urn:xmpp:mix:1</value>
        </field>
        <field var='Name'>
            <value>Witches Coven</value>
        </field>
        <field var='Description'>
           <value>A location not far from the blasted heath where
                  the three witches meet</value>
         </field>
         <field var='Contact'>
            <value>greymalkin@shakespeare.example</value>
         </field>
         <field var='JID_Visibility'>
            <value>jid-visible</value>
         </field>
      </x>
  </item>
</items>
```

3.9.9 Allowed

This node represents a list of JIDs that are allowed to become participants. If the Allowed node is not present, all JIDs are allowed. This node is accessed and managed using standard pubsub. The Allowed list is always considered in conjunction with the banned list, stored in the banned node. Only Administrators and Owners have write permission to the Allowed node and are also the only roles that are allowed to subscribe to this node. The Allowed node is a permanent node. Each item contains a real bare JID. The following example shows how the Allowed list can specify single JIDs and domains.

Listing 5: Allowed Node

```
<items node='urn:xmpp:mix:nodes:allowed'>
  <item id='shakespeare.example'/>
  <item id='alice@wonderland.example'/>
</items>
```

3.9.10 Banned

This node represents a list of JIDs that are explicitly not allowed to become participants. The values in this list take priority over values in the Allowed node. This node is accessed and managed using standard pubsub Only Administrators and Owners have write permission to the Banned node and are also the only roles that are allowed to subscribe to this node. Each item contains a real bare JID. The Banned node can contain bare JIDs and/or domains. The Banned node is a permanent node.

Listing 6: Banned Node

```
<items node='urn:xmpp:mix:nodes:banned'>
  <item id='lear@shakespeare.example'/>
  <item id='macbeth@shakespeare.example'/>
</items>
```

3.9.11 Configuration Node

The Configuration node holds the configuration of the channel as a single item, named by the date-time of the last update to the configuration. The Configuration node is a permanent node with a maximum of one item. Previous configuration history MAY be accessed by MAM. Users with read access to the configuration node MAY subscribe to the configuration node to get notification of configuration change. This node is accessed and managed using standard pubsub. The configuration node is OPTIONAL for a MIX channel. For example, configuration choices could be fixed and not exposed. A subset of the defined configuration options MAY be used and additional non-standard configuration options MAY be added. JIDs in the configuration MUST be real bare JIDs and not proxy JIDs. If configuration options to control functionality of the nature described here are provided, the options defined in this standard MUST be used. The following configuration attributes are defined:

Name	Description	Field Type	Values	Default
'Last Change Made By'	Bare JID of the user making the last change.	jid-single	-	-

Name	Description	Field Type	Values	Default
'Owner'	Bare JIDs with Owner rights as defined in ACL node. When a channel is created, the JID creating the channel is configured as an owner, unless this attribute is explicitly configured to another value.	jid-multi	-	-

Name	Description	Field Type	Values	Default
'Administrator'	Bare JIDs with Administrator rights.	jid-multi	-	-
'End of Life'	The date and time at which the channel will be automatically removed by the server. If this is not set, the channel is permanent.	text-single	-	-

Name	Description	Field Type	Values	Default
'Nodes Present'	Specifies which nodes are present. Presence of config nodes is implicit. Jidmap node MUST be present if participants node is present. 'avatar' means that both Avatar Data and Avatar Metadata nodes are present.	list-multi	'participants'; 'presence'; 'information'; 'allowed'; 'banned'; 'jidmap-visible'; 'avatar'	-

Name	Description	Field Type	Values	Default
'Messages Node Subscription'	Controls who can subscribe to messages node.	list-single	'participants'; 'allowed'; 'anyone'	'participants'
'Presence Node Subscription'	Controls who can subscribe to presence node.	list-single	'participants'; 'allowed'; 'anyone'	'participants'
'Participants Node Subscription'	Controls who can subscribe to participants node.	list-single	'participants'; 'allowed'; 'anyone'; 'nobody'; 'admins'; 'owners'	'participants'
'Information Node Subscription'	Controls who can subscribe to the information node.	list-single	'participants'; 'allowed'; 'anyone'	'participants'

Name	Description	Field Type	Values	Default
'Allowed Node Subscription'	Controls who can subscribe to allowed node.	list-single	'participants'; 'allowed'; 'nobody'; 'admins'; 'owners'	'admins'
'Banned Node Subscription'	Controls who can subscribe to banned node.	list-single	'participants'; 'allowed'; 'nobody'; 'admins'; 'owners'	'admins'
'Configuration Node Access'	Controls who can subscribe to configuration node and who has read access to it.	list-single	'participants'; 'allowed'; 'nobody'; 'admins'; 'owners'	'owners'

Name	Description	Field Type	Values	Default
'Information Node Update Rights'	Controls who can make changes to the information node	list-single	'participants'; 'admins'; 'owners'	'admins'
'Avatar Nodes Update Rights'	Controls who can make changes to the avatar data and metadata nodes	list-single	'participants'; 'admins'; 'owners'	'admins'

Name	Description	Field Type	Values	Default
'Open Presence'	If selected, any client MAY register presence. If not selected, only clients with bare JID in the participants list are allowed to register presence.	boolean	-	false

Name	Description	Field Type	Values	Default
'Participants Must Provide Presence'	If selected, all channel participants are REQUIRED to share presence information with the channel.	boolean	-	false

Name	Description	Field Type	Values	Default
'User Message Retraction'	If this option is selected users will be able to retract messages that they have sent to the MIX channel.	boolean	-	false
'Administrator Message Retraction Rights'	This controls which group is able to retract any message sent to the MIX channel.	list-single	'nobody';'owners' 'admins'; 'owners'	

Name	Description	Field Type	Values	Default
’Participation Addition by Invitation from Participant’	This option extends a channel so that a channel participant has rights to invite and enable other users as participants.	boolean	-	false

Name	Description	Field Type	Values	Default
'Private Messages'	If this option is selected, private messages MAY be used with the channel.	boolean	-	true

The configuration node is in Data Forms (XEP-0004) [14] format and includes all of the options used by the channel, including values for options using default values. This means that the value in the form can be directly mapped with the form returned by configuration administration commands. Configuration nodes will typically have a large number of elements. The following short example is provided to illustrate the syntax of the configuration node.

Listing 7: Configuration Node

```
<items node='urn:xmpp:mix:nodes:config'>
  <item id='2016-05-30T09:00:00'>
      <x xmlns='jabber:x:data' type='result'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <field var='Owner'>
            <value>hecate@shakespeare.example</value>
        </field>
        <field var='Owner'>
            <value>greymalkin@shakespeare.example</value>
        </field>
        <field var='Messages Node Subscription'>
           <value>allowed</value>
         </field>
         <field var='No Private Messages'>
```

[14] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

```
            <value>true</value>
          </field>
          </x>
  </item>
</items>
```

3.10 Non-Standard Nodes

The MIX standard allows channels to use non-standard nodes, using names that do not conflict with the standard nodes. Non-Standard nodes MUST NOT duplicate or otherwise provide capability that can be provided by standard nodes.

4 Error Handling

The MIX specification is built on layered services that have defined errors. This enables the core MIX specification to reflect primarily the successful use case, as in almost all cases the error reporting of the layer service provides what is needed. A message sender MUST be prepared to handle any valid error from the layer services. When a message receiver encounters an error situation, it MUST use the most appropriate layer server error to report this issue back to the sender. For example a receiving entity might use the ”not authorized” error in response to a disco query that is not authorized. Errors for the following layer services need to be handled for MIX:

1. IQ. All of the IQ errors of RFC 6120 [15] MUST be supported.
2. Messages. If a message is received and an error situation is encountered, a message of type error MUST be sent back to the message sender. This message format is specified in RFC 6121 [16] containing an error defined in RFC 6120 [17], which is the same error set as for IQs.
3. Presence. Any responses to presence messages MUST follow the rules of RFC 6121 [18].
4. PubSub. Where MIX protocol messages use PubSub protocol, the errors defined in Publish-Subscribe (XEP-0060) [19] MUST be used and supported.

[15] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[16] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[17] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[18] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[19] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.

5 Discovery

5.1 Discovering a MIX service

An entity MAY discover a MIX service or MIX services by sending a Service Discovery items ("disco#items") request to its own server.

Listing 8: Entity queries Server for associated services

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='lx09df27'
    to='shakespeare.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>

<iq from='shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='mix.shakespeare.example'
          name='Shakespearean Chat Service'/>
    <item jid='mix2.shakespeare.example'
          name='Another Shakespearean Chat Service'/>
  </query>
</iq>
```

To determine if a domain hosts a MIX service, a Service Discovery (XEP-0030) [20] info query is sent in the usual manner to determine capabilities.

Listing 9: Entity queries a service

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

The MIX service then MUST return its identity and the features it supports, which MUST include the 'urn:xmpp:mix:1' feature, and the identity MUST have a category of 'conference' and a type of 'text', as shown in the following example:

Listing 10: Service responds with Disco Info result

```
<iq from='mix.shakespeare.example'
```

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

```
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='Shakespearean Chat Service'
        type='text'/>
    <feature var='urn:xmpp:mix:1'/>
    <feature var='urn:xmpp:mix:1#searchable'>
  </query>
</iq>
```

A MIX service MUST return the 'urn:xmpp:mix:1' feature and MAY return the other features listed here:

- 'urn:xmpp:mix:1': This indicates support of MIX, and is returned by all MIX services.
- 'urn:xmpp:mix:1#searchable': This is shown in the above example and indicates that a the MIX Service MAY be searched for channels. This presence of this feature can be used by a client to guide the user to search for channels in a MIX service.
- 'urn:xmpp:mix:1#create-channel': This is described in Checking for Permission to Create a Channel in support of channel administration. When an end user client needs to create channels, perhaps for short term usage, this feature helps the client to identify an MIX service to use. It enables a configuration where permanent (searchable) channels are placed in one MIX service and clients will be able to create channels in another MIX service which is not searchable.

A MIX service MUST NOT advertise support for Message Archive Management (XEP-0313) [21], as MAM is supported by the channels and not by the service. A MIX service MUST NOT advertise support for generic Publish-Subscribe (XEP-0060) [22], as although MIX makes use of PubSub it is not a generic PubSub service.

5.2 Discovering the Channels on a Service

The list of channels supported by a MIX service is obtained by a disco#items command. The MIX service MUST only return channels that exist and that the user making the query has rights to subscribe to.

Listing 11: Client Queries for Channels on MIX Service

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
```

[21] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
[22] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.

```
    id='kl2fax27'
    to='mix.shakespeare.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

Listing 12: MIX Service Returns Disco Items Result

```
<iq from='mix.shakespeare.example'
    id='kl2fax27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='coven@mix.shakespeare.example' />
    <item jid='spells@mix.shakespeare.example' />
    <item jid='wizards@mix.shakespeare.example' />
  </query>
</iq>
```

5.3 Discovering Channel Information

In order to find out more information about a given channel, a user can send a disco#info query to the channel.

Listing 13: Entity Queries for Information about a Specific Channel

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='ik3vs715'
    to='coven@mix.shakespeare.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info' node='mix'/>
</iq>
```

If the querying user is allowed to subscribe, the channel MUST return its identity and the features it supports. Note that a MIX channel MUST support MAM and so the response will always include both MIX and MAM support. All disco queries on a MIX channel and results returned MUST include the attribute node='mix'. The reason for this is to facilitate MIX channels and Multi-User Chat (XEP-0045) [23] MUC rooms sharing the same JID. This extra parameter will enable a server to return appropriate information.

Listing 14: Channel Returns Disco Info Result

```
<iq from='coven@mix.shakespeare.example'
    id='ik3vs715'
    to='hag66@shakespeare.example/UUID-c8y/1573'
```

[23] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

```
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info' node='mix'>
    <identity
        category='conference'
        name='A␣Dark␣Cave'
        type='mix'/>
    <feature var='urn:xmpp:mix:1'/>
    <feature var='urn:xmpp:mam:1'/>
  </query>
</iq>
```

5.4 Discovering Nodes at a Channel

Use disco#items to find the nodes associated with a channel. Discovering nodes in MIX MUST use the node='mix' attribute. The MIX service MUST only return nodes that exist and that the user making the query has rights to subscribe to.

Listing 15: Entity Queries for Nodes at a Channel

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='kl2fax27'
    to='coven@mix.shakespeare.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/ node='mix'>
</iq>
```

Listing 16: Channel Returns Disco Items Result

```
<iq from='coven@mix.shakespeare.example'
    id='kl2fax27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items' node='mix'>
    <item jid='coven@mix.shakespeare.example'
          node='urn:xmpp:mix:nodes:presence'/>
    <item jid='coven@mix.shakespeare.example'
          node='urn:xmpp:mix:nodes:participants'/>
    <item jid='coven@mix.shakespeare.example'
          node='urn:xmpp:mix:nodes:messages'/>
    <item jid='coven@mix.shakespeare.example'
          node='urn:xmpp:mix:nodes:config'/>
  </query>
</iq>
```

5.5 Determining Information about a Channel

The Information Node contains various information about the channel that can be useful to the user, that the client can access using PubSub, as shown below.

Listing 17: Client Requests Channel Information

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='kl2fax27'
    to='coven@mix.shakespeare.example'
    type='get'>
  <pubsub xlns='http://jabber.org/protocol/pubsub'>
      <items node='urn:xmpp:mix:nodes:info'/>
  </pubsub>
</iq>
```

Listing 18: MIX Service Returns Channel Information

```
<iq from='coven@mix.shakespeare.example'
    id='kl2fax27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <pubsub xlns='http://jabber.org/protocol/pubsub'>
      <items node='urn:xmpp:mix:nodes:info'>
           <item id='2016-05-30T09:00:00' xmlns='urn:xmpp:mix:1'>
              <x xmlns='jabber:x:data' type='result'>
                 <field var='FORM_TYPE' type='hidden'>
                     <value>urn:xmpp:mix:1</value>
                 </field>
                 <field var='Name'>
                     <value>Witches Coven</value>
                 </field>
                 <field var='Description'>
                    <value>A location not far from the blasted heath
                       where
                          the three witches meet</value>
                  </field>
                  <field var='Contact'>
                       <value>greymalkin@shakespeare.example</value>
                  </field>
               </x>
         </item>
      </items>
   </pubsub>
</iq>
```

5.6 Determining the Participants in a Channel

Participants in the channel are determined using PubSub retrieval of the Participants Node which will give proxy JID and nick.

Listing 19: User's Client Requests Participant List

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='kl2fax27'
    to='coven@mix.shakespeare.example'
    type='get'>
     <pubsub xlns='http://jabber.org/protocol/pubsub'>
          <items node='urn:xmpp:mix:nodes:participants'/>
     </pubsub>
</iq>
```

Listing 20: MIX Service Returns Participant List

```
<iq from='coven@mix.shakespeare.example'
    id='kl2fax27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
     <pubsub xlns='http://jabber.org/protocol/pubsub'>
          <items node='urn:xmpp:mix:nodes:participants'>
             <item id='123456#coven@mix.shakespeare.example'>
                 <participant xmlns='urn:xmpp:mix:1'>
                      <nick>thirdwitch</nick>
                  </participant>
             </item>
              <item id='87123#coven@mix.shakespeare.example'>
                 <participant xmlns='urn:xmpp:mix:1'>
                      <nick>top witch</nick>
                  </participant>
             </item>
           </items>
     </pubsub>
  <items>
</iq>
```

Online clients in the channel can be looked up in a similar manner by PubSub query of the Presence node.

5.7 Discovering Client MIX Capability

Where a client supports MIX, it MUST advertise this capability in response to a Disco request. This will enable other entities to determine if a client supports MIX, and in particular it facilitates the client's server to determine client support. This can be optimized by use of CAPS. The following example shows a Disco request to and response from a client that supports both MIX and MUC.

Listing 21: Disco Query for MIX support

```
<iq  from='juliet@capulet.example/UUID-e3r/9264'
```

```
    id='d1rt87mr4w'
    to='romeo@montague.example/UUID-m2t/3945'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>


<iq from='romeo@montague.example/UUID-m2t/3945'
    id='d1rt87mr4w'
    to='juliet@capulet.example/UUID-e3r/9264'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='http://jabber.org/protocol/caps'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <feature var='urn:xmpp:mix:1'/>
  </query>
</iq>
```

6 Use Cases

6.1 Common User Use Cases

6.1.1 Joining a Channel

A user joins a channel by sending a MIX ”join” command from one of the user’s clients. There is no default set of nodes, so the user MUST specify the set of nodes to be subscribed to. To achieve the equivalent service to MUC, a user would subscribe to both messages and presence nodes. A user will typically subscribe to at least the message and/or presence nodes but MAY join and not subscribe to any nodes. The <join/> is a child element of <iq/> element. The <join/> element is qualified by the ’urn:xmpp:mix:1’ namespace. The channel is specified by a ’channel’ attribute in the <join/> element. The requested nodes are encoded as <subscribe/> child elements of the <join/> element. The join leads to the server subscribing the user to each of the requested nodes associated with the channel. The MIX service will also add the user to the participant list by injecting a new item into the ”urn:xmpp:mix:nodes:participants” node automatically.
The default MIX model is that only channel participants are allowed to subscribe to nodes. A MIX channel MAY allow non-participant subscription. This will be handled by clients directly subscribing to the desired PubSub nodes.
The user’s server needs to make roster changes as part of the join functionality. Because of this, the join and leave operations need to operate indirectly. For this reason the initial join request is sent to the local server with the MIX channel specified as an attribute to the join. The join is sent from a client identified by a full JID to the user’s bare JID.

Listing 22: Client sends request to Local Server to Join a Channel

```
<iq type='set'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='hag66@shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1'
        channel='coven@mix.shakespeare.example'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <subscribe node='urn:xmpp:mix:nodes:presence'/>
    <subscribe node='urn:xmpp:mix:nodes:participants'/>
    <subscribe node='urn:xmpp:mix:nodes:config'/>
  </join>
</iq>
```

The user's server will then pass the request to the MIX channel, with the request coming from the user's bare JID.

Listing 23: User's Server sends Join request to MIX Channel

```
<iq type='set'
    from='hag66@shakespeare.example'
    to='coven@mix.shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <subscribe node='urn:xmpp:mix:nodes:presence'/>
    <subscribe node='urn:xmpp:mix:nodes:participants'/>
    <subscribe node='urn:xmpp:mix:nodes:config'/>
  </join>
</iq>
```

The channel responds to the user's sever with an IQ-result addressed to the user's bare JID. This stanza includes the nodes to which the user has been successfully subscribed, as well as the bare JID that will be used for the user in this channel and added to the participant list. The JID returned in the join is the user's bare proxy JID. The following example shows the result of the above request when the request is completed in full.

Listing 24: Channel responds to User's Server

```
<iq type='result'
    from='coven@mix.shakespeare.example'
    to='hag66@shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1' jid='123456#coven@mix.shakespeare.
      example'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <subscribe node='urn:xmpp:mix:nodes:presence'/>
    <subscribe node='urn:xmpp:mix:nodes:participants'/>
```

```
    <subscribe node='urn:xmpp:mix:nodes:config'/>
  </join>
</iq>
```

The user's server will then make roster modifications as set out in the next section. After making these changes, the user's server will send the join response back to the client that made the join request, identified by the full JID. This is illustrated below:

Listing 25: User's Server sends response to Client

```
<iq type='result'
    from='hag66@shakespeare.example'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1' jid='123456#coven@mix.shakespeare.
      example'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <subscribe node='urn:xmpp:mix:nodes:presence'/>
    <subscribe node='urn:xmpp:mix:nodes:participants'/>
    <subscribe node='urn:xmpp:mix:nodes:config'/>
  </join>
</iq>
```

This response informs the client that initiated the join request that the request has been completed. Note that the roster changes described in the next section will lead to roster update information being sent to all of the user's online clients, so that all of the user's clients will be updated with the new MIX channel information.
If a user cannot be subscribed to one or more of the requested nodes (e.g., because the node does not exist), but can be subscribed to some the response simply lists the nodes successfully subscribed. If at least one node is requested and none of the nodes requested are successfully subscribed to, an error response is sent indicating the reason that the first node requested was not subscribed to. This error response will also include other nodes requested where subscription failed for the same reason.
The following response example shows a successful response to the initial request example where the participant is not subscribed to all nodes associated with the channel (in this case only messages, participants, and information). This example shows the message from the MIX channel to the user's server, which will be modified and sent to the client.

Listing 26: Channel Processes Join With Some Nodes Not Subscribed To

```
<iq type='result'
    from='hag66@shakespeare.example'
    to='coven@mix.shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1' jid='123456#coven@mix.shakespeare.
      example'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
```

```
    <subscribe node='urn:xmpp:mix:nodes:participants'/>
    <subscribe node='urn:xmpp:mix:nodes:info'/>
  </join>
</iq>
```

The channel also adds the user to the participants node and sends a notification to subscribers of the participants node. The following example shows such a notification.

Listing 27: Channel Adds User to Participants Node

```
<message from='coven@mix.shakespeare.example'
         to='hecate@shakespeare.example'
         id='5A9C7398-DB13-4BFA-A091-2D466C710AAF'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='urn:xmpp:mix:nodes:participants'>
      <item id='123456#coven@mix.shakespeare.example'>
        <participant xmlns='urn:xmpp:mix:1'/>
      </item>
    </items>
  </event>
</message>
```

The user that has been added to the channel is identified by the item id of the item added to the Participants node, which is the proxy JID of the new channel participant. Note that the <participant> element MUST NOT include a nick of the user being added. The nick MAY be set after the join.
At the same time the participant MUST be added to the JID Map node, to map from proxy JID to real JID. For a JID Maybe Visible channel, the participant MUST be added to the JID Maybe Visible Map node. The value in this node MUST reflect the user's visibility preference for the channel and MUST be updated to reflect any changes to this preference.
A user MAY subsequently modify subscription to nodes in a channel by sending a subscription modification request encoded as a <update-subscription/$gt; child element of <iq/> element. The <update-subscription/$gt; element is qualified by the 'urn:xmpp:mix:1' namespace. The requested notes are encoded as <subscribe/> child elements of the <update-subscription/$gt; element with the node name encoded as a 'node' attribute. This modification goes directly from client to MIX channel, as this change does not impact the roster and so does not need any local action. The following example shows subscription modification.

Listing 28: User Modifies Subscription Request

```
<iq type='set'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='coven@mix.shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <update-subscription xmlns='urn:xmpp:mix:1'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
  </update-subscription>
```

```
</iq>

<iq type='result'
     from='coven@mix.shakespeare.example'
     to='hag66@shakespeare.example/UUID-a1j/7533'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <update-subscription xmlns='urn:xmpp:mix:1' jid='hag66@shakespeare.
      example'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
  </update-subscription>
</iq>
```

6.1.2 Roster Management

As part of the channel joining process, the user's server MUST add the MIX channel to the user's roster. This is done to share the user's presence status with the channel and channel participants subscribing to presence, when the user wishes this presence to be shared. These roster entries also enables the user's client to quickly determine which channels the user has joined. This roster entry will lead to the user's server correctly sending user's presence from all the user's MIX clients to the MIX channel. Where the user wishes to share presence, the roster subscription is configured with one way presence, as presence is sent to the MIX channel but no presence information is sent about the MIX channel to the user.
If the user does not wish to publish presence information to the channel, the user's server will add the roster entry with mode subscription=none. The roster entry will be present to record that the user has joined the channel, but it will not send presence information to the channel. The user's server MUST do this when the user has chosen Presence preference of 'not share' as described below. If the user changes the value of the preference, the server MUST modify subscription mode to reflect this.
The user's server MUST remove this roster entry when the user leaves the channel.
A channel MAY publish an Avatar following User Avatar (XEP-0084) [24]. A client MAY make use of this information to associate an Avatar with the roster entry for a channel.

6.1.3 User Preferences and Additional Information

A channel MAY store user preferences and parameters with each user. A user JID visibility preference has the following values:

1. 'default'. In this setting, the channel default value will be used. This value is used if another value is not explicitly set. This means JID is visible for a JID Visible channel and JID is hidden for JID Hidden and JID Maybe Visible channels.
2. 'never'. If this is set, JID will never be shared and user will be removed from the channel if its mode changes to JID Visible.

[24] XEP-0084: User Avatar <https://xmpp.org/extensions/xep-0084.html>.

3. 'always'. If this is set, JID will always be shared and users will be removed from the channel if its mode changes to JID Hidden.

4. 'prefer not'. If this is set, JID will only be shared if mode is JID Visible.

The user MAY specify that no Private Messages are to be sent from the channel, and allow vCard retrieval.
The user MAY specify that presence is not to be shared, which will prevent the MIX Channel from sending a presence probe for the user on channel start-up. The user will also choose to not include the MIX channel in the user's roster, so that presence will not be updated. Where the channel configuration sets 'Participants Must Provide Presence', this variable MUST be set to 'Share'.
The following table sets out the standardized user preference values. A MIX implementation MAY use other values.

Option	Values	Default
'JID Visibility'	'default', 'never', 'always', 'prefer not'	'default'
'Private Messages'	'allow', 'block'	'allow'
'vCard'	'allow', 'block'	'block'
'Presence'	'share', 'not share'	'share'

When joining a channel, the client MAY specify one or more preference options as a Data Forms (XEP-0004) [25] form. In the response, the server MUST specify all of the user preferences supported by the server, with default values if the user has not requested a different value. The following example shows joining a channel and setting a preference. The following example shows the message sent from the user's server to the MIX channel, which will have been preceded by a message from the user's client to the user's server.

Listing 29: User Joins a Channel and Specifies a preference

```
<iq type='set'
    from='hag66@shakespeare.example'
    to='coven@mix.shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
    <join xmlns='urn:xmpp:mix:1'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <subscribe node='urn:xmpp:mix:nodes:presence'/>
    <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
```

[25] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

```
        <field var='JID_Visibility'>
            <value>never</value>
        </field>
    </x>
  </join>
</iq>
```

The following example shows the result of a successful join, which also reports all the user preferences. This example shows the message coming from the MIX channel to the user's server, which will be sent on to the user.

Listing 30: Channel Successfully Processes Join and returns the preferences set

```
<iq type='result'
    from='coven@mix.shakespeare.example'
    to='hag66@shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1' jid='hag66@shakespeare.example'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <subscribe node='urn:xmpp:mix:nodes:presence'/>
    <x xmlns='jabber:x:data' type='result'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <field var='JID_Visibility'>
            <value>never</value>
        </field>
        <field var='Private_Messages'>
           <value>allow</value>
         </field>
        <field var='vCard'>
          <value>block</value>
         </field>
    </x>
  </join>
</iq>
```

The client MAY also query the channel in order to find out which user preferences are supported and the options available. This will allow users to set options not specified in the standard, by providing a form template in the result. The request is encoded as a <user-preference/> child element of <iq/>. <user-preference/> is qualified by the 'urn:xmpp:mix:1' namespace. The result is encoded as a form child element in the <user-preference/> element.

Listing 31: User Requests and Recieves Preferences Template Form

```
<iq type='get'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='coven@mix.shakespeare.example'
```

```
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
    <user-preference xmlns='urn:xmpp:mix:1'/>
</iq>

<iq type='result'
    from='coven@mix.shakespeare.example'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
    <user-preference xmlns='urn:xmpp:mix:1'>
    <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <field type='list-single' label='Preference␣for␣JID␣Visibility
            '
                    var='JID␣Visibility'>
            <option label='JID␣Never␣Shown'><value>Never</value></
                option>
            <option label='Default␣Behaviour'>
                    <value>default</value></option>
            <option label='Try␣not␣to␣show␣JID'>
                     <value>prefer not</value></option>
         </field>
         <field type='list-single' label='Example␣Custom␣Preference'
                 var='Custom␣Preference'>
            <option label='One␣Option'><value>a</value></option>
            <option label='Another␣Option'><value>b</value></option>
         </field>
     </x>
  </user-preference>
</iq>
```

A user MAY modify preferences using the corresponding set operation. The set MAY specify values for some or all attributes. All attributes are returned in the result.

Listing 32: User Updates Preferences

```
<iq type='set'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='coven@mix.shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
    <user-preference xmlns='urn:xmpp:mix:1'/>
     <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <field var='JID␣Visibility'>
            <value>never</value>
        </field>
```

```
        <field var='Private_Messages'>
           <value>allow</value>
         </field>
        <field var='vCard'>
          <value>block</value>
         </field>
     </x>
</iq>

<iq type='result'
    from='coven@mix.shakespeare.example'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
    <user-preference xmlns='urn:xmpp:mix:1'>
    <x xmlns='jabber:x:data' type='result'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <field var='JID_Visibility'>
            <value>never</value>
        </field>
        <field var='Private_Messages'>
           <value>allow</value>
         </field>
        <field var='vCard'>
          <value>block</value>
         </field>
     </x>
  </user-preference>
</iq>
```

6.1.4 Leaving a Channel

Users generally remain in a channel for an extended period of time. In particular the user remains as a participant the channel when the user goes offline. Note that this is different to Multi-User Chat (XEP-0045) [26], where the client leaves a room when going offline. So, leaving a MIX channel is a permanent action for a user across all clients. In order to leave a channel, a user sends a MIX "leave" command to the channel. The leave command is encoded as a <leave/> child element of <iq/> element. The <leave/> element is qualified by the 'urn:xmpp:mix:1' namespace, with the channel specified as a 'channel" attribute. When a user leaves the channel, the user's server will remove the channel from the user's roster. Leave commands are sent indirectly through the user's server, to enable roster removal. Leaving is initiated by a client request, as shown in the following example.

[26] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

Listing 33: Client Requests to Leave a Channel

```
<iq type='set'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='hag66@shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <leave xmlns='urn:xmpp:mix:1'
         channel='coven@mix.shakespeare.example'/>
</iq>
```

The user's server will then generate a matching request to the MIX channel.

Listing 34: User's Server sends Leave Request to a Channel

```
<iq type='set'
    from='hag66@shakespeare.example'
    to='coven@mix.shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <leave xmlns='urn:xmpp:mix:1'/>
</iq>
```

The MIX channel will then remove the user from the channel, as described below. A response is sent to the user's server.

Listing 35: Channel Confirms Leave to User's Server

```
<iq type='result'
    from='coven@mix.shakespeare.example'
    to='hag66@shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <leave xmlns='urn:xmpp:mix:1'/>
</iq>
```

After receiving the confirmation that the user has left the MIX channel, the user's server will remove the MIX channel entry from the user's roster. The user's server will then notify the client that requested to leave.

Listing 36: User's Server Confirms Leave to Client

```
<iq type='result'
    from='hag66@shakespeare.example'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <leave xmlns='urn:xmpp:mix:1'/>
</iq>
```

When the user leaves the channel, the MIX service is responsible for unsubscribing the user from all nodes in the channel and for removing the user from the participants and presence list. If the user has online presence when the user leaves the channel, the change of presence

status caused by removing the user's entry or entries from the presence node will ensure that subscribers to the presence node are correctly updated on presence status. Deletion from the participants and presence functions as if the item (channel participant) had been deleted using the PubSub retract mechanism with notification set to true. Notification of the deletion is sent to clients subscribed to the participants PubSub nodes, as shown in the example below.

Listing 37: Reporting when User Leaves a Channel

```
<message from='coven@mix.shakespeare.example'
                 to='hecate@shakespeare.example' id='f5pp2toz'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='urn:xmpp:mix:nodes:participants'>
      <retract id='123456#coven@mix.shakespeare.example'/>
    </items>
  </event>
</message>

<message from='coven@mix.shakespeare.example'
                 to='other-witch@shakespeare.example' id='bar'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='urn:xmpp:mix:nodes:presence'>
      <retract id='123456#coven@mix.shakespeare.example/8765'/>
    </items>
  </event>
</message>
```

6.1.5 Setting a Nick

Each participant of a channel MAY have a nick, which is how other users in the channel will see the user. In some cases a nick is not needed, for example where a user reads messages in a channel but does not send messages or share presence information. A nick MUST be present for a user to send a message to a channel or for a user's presence to be shared on a channel. There are four ways that a user's nick can be obtained. The choice of mechanism or mechanisms is dependent on channel policy:

1. The nick is registered with the user account in some way, for example as part of user provisioning with nick configured as an attribute in a directory service. For example, this could be chosen by corporate services that wish to ensure consistent nick values for a set of users and channels.
2. The nick is registered with the MIX service, as described in Registering a Nick .
3. The user explicitly sets the nick, as described in this section.
4. The MIX service generates the nick. In this case it is RECOMMENDED that the assigned

nick is a UUID following RFC 4122 [27].

A user will typically set a nick when joining a channel and MAY update this nick from time to time. The user does this by sending a command to the channel to set the nick. This command is a <setnick/> child element of <iq/> element. The <setnick/> element is qualified by the 'urn:xmpp:mix:1' namespace. The nick is encoded as a <nick/> child element of the <setnick/> element. If the user wishes the channel to assign a nick (or knows that the channel will assign a nick) the nick field can be left blank, so that the user can see what is assigned in the result.

Listing 38: User sets Nick on Channel

```
<iq type='set'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='coven@mix.shakespeare.example'
    id='7nve413p'>
  <setnick xmlns='urn:xmpp:mix:1'>
    <nick>thirdwitch</nick>
  </setnick>
</iq>
```

On successful nick assignment, the channel will return the nick that is to be used, noting that this MAY be different to the requested nick. MIX services SHOULD apply the "nickname" profile of the PRECIS OpaqueString class, as defined in RFC 7700 [28]. The channel MAY return a conflict error or other appropriate error.

Listing 39: Channel informs user of Nick

```
<iq type='result'
    from='coven@mix.shakespeare.example'
    to'hag66@shakespeare.example/UUID-a1j/7533'
    id='7nve413p'>
  <setnick xmlns='urn:xmpp:mix:1'>
    <nick>thirdwitch</nick>
  </setnick>
</iq>
```

6.1.6 Registering a Nick

A nick MAY be associated with the user's bare JID. A user can register a nick with the MIX service. Nick registration can be used ensure that a user is able to use the same nick in all channels in the service and to prevent other users from using a registered nick. This can help achieve a consistent experience across a set of channels and prevent user confusion. Support for nick registration by a MIX service is OPTIONAL. Where nick registration is supported, nick

[27] RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace <http://tools.ietf.org/html/rfc4122>.
[28] RFC 7700: Preparation, Enforcement, and Comparison of Internationalized Strings Representing Nicknames<http://tools.ietf.org/html/rfc7700>.

registration MAY be OPTIONAL or MANDATORY. Where a user has registered a Nick with the MIX service, it MAY be used by each channel according to policy for the channel. A channel MAY enforce use of a registered nick. A channel MUST NOT use a registered nick for any other participant.
In order to determine if a Nick is allowed to be registered, the user MAY use disco to determine capabilities of the MIX service.

Listing 40: User Determines features of the MIX service

```
<iq type='get'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='mix.shakespeare.example'
    id='7nve413p'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq type='result'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    from='mix.shakespeare.example'
    id='7nve413p'>
    <query xmlns='http://jabber.org/protocol/disco#info'/>
        <feature var='urn:xmpp:mix:1#nick-register'/>
    </query>
</iq>
```

The response will be a list of features of the MIX channel. If Nick Registration is supported, then the result set will include <feature var="urn:xmpp:mix:1#nick-register"/>.
To register a nick with the MIX service the user sends a register command to the service. This is encoded as a <register/> child element of an <iq/> element. The <register/> element is qualified by the urn:xmpp:mix:1' namespace. The nick is encoded in a <nick/> child element of the <register/> element.

Listing 41: User Registers with Service

```
<iq type='set'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='mix.shakespeare.example'
    id='7nve413p'>
  <register xmlns='urn:xmpp:mix:1'>
    <nick>thirdwitch</nick>
  </register>
</iq>
```

On success, the service informs the user of its nick. MIX SHOULD apply the "nickname" profile of the PRECIS OpaqueString class, as defined in RFC 7700 [29] to the requested nick. This means

[29] RFC 7700: Preparation, Enforcement, and Comparison of Internationalized Strings Representing Nicknames<http://tools.ietf.org/html/rfc7700>.

that nick that is issued might be different from the nick that was requested.
When an Nick is assigned, the MIX server MUST update the Participants Node in the channel to reflect this change. Any users subscribed to this node will be notified of the change of Nick. The following example shows an example of reporting successful Nick assignment.

Listing 42: Service Returns User of Nick

```
<iq type='result'
    to='mix.shakespeare.example'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    id='7nve413p'>
  <register xmlns='urn:xmpp:mix:1'>
    <nick>thirdwitch</nick>
  </register>
</iq>
```

If the requested nick is already taken and the MIX service does not assign an alternate nick, the MIX service MUST return a <conflict/> error:

Listing 43: Nick is Taken

```
<iq type='error'
    to='mix.shakespeare.example'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    id='7nve413p'>
  <error type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the register request does not contain a <nick/> element, then the MIX service MUST assign one. It is RECOMMENDED that the assigned nick is a UUID following RFC 4122 [30].

6.1.7 Setting User Presence

A user joins a channel over an extended period, and participation in a channel does not generally change when user goes online or offline. The user's participation in a channel is reflected by the user's bare JID in the participant node. All messages to the channel are sent to this JID.
A user MAY share presence information with the channel, for one or more online clients. Where a user shares presence information with a channel, the channel is entered by the user's server into the user's roster when the user subscribes to the channel. When an XMPP client comes online or changes presence status, this will be communicated by the user to the user's server using broadcast presence. The user's XMPP server is then responsible to share this

[30] RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace <http://tools.ietf.org/html/rfc4122>.

presence information to each entry in the roster and in particular to each MIX channel in the roster. The MIX channel will update the ”urn:xmpp:mix:nodes:presence” node with any change of status and the updated presence information and then share this updated presence with users subscribed to this node, as described below. When the user sets an explicit status, this is used to modify the presence node in the channel. When a client being used by the user goes offline, the associated server will send presence status ”unavailable” to the MIX channel, which will cause the full JID for that client to be removed from the presence node.
A channel MAY require that all channel participants share presence information with the channel, which is represented in the ”urn:xmpp:mix:nodes:presence” node. If sharing presences is needed by the channel, an XMPP client conforming to this specification MUST share presence with the channel by including the channel in the user’s roster. Note that the MIX service cannot generally enforce this, but it can require and enforce that when a message is sent to the channel, that the sender of the message is in the presence list.
Presence status and availability is set in a MIX channel by standard presence stanzas sent to the MIX channel by the user’s server. Users wishing to receive presence updates will subscribe to the MIX channel presence node. Presence updates are sent out to subscribing participants using standard presence stanzas.
A user setting status is now used as an example. Unlike in Multi-User Chat (XEP-0045) [31] where coming online is a special action, coming online in MIX is implicit when presence status is set. Going offline is a achieved by setting presence status to unavailable, which removes the client full JID entry from the presence node. When a user sets a presence status, the user’s server sends updated presence to the MIX channel, and the MIX service then publishes the user’s availability to the ”urn:xmpp:mix:nodes:presence” node. If there is not an item named by the full JID of the client with updated presence status, this item is created. The sequence is shown in the following examples, starting with a client setting presences status on the connected server.

Listing 44: Client Sets Presence Status on Server

```
<presence xmlns='jabber:client' from‘=hag66@shakespeare.example/UUID-
    a1j'/7533>
  <show>dnd</show>
  <status>Making a Brew</status>
</presence>
```

The server then sends the presence information to roster entries. The following example then shows the presence message from the client’s server to the MIX channel.

Listing 45: Server sends Presence Status to MIX Channel

```
<presence  from‘=hag66@shakespeare.example/UUID-a1j'/7533
             to='coven@mix.shakespeare.example'>
  <show>dnd</show>
  <status>Making a Brew</status>
```

[31] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

```
</presence>
```

The user's presence information is then published by the service to the "urn:xmpp:mix:nodes:presence" node, with the 'publisher' attribute set to the user's participant identifier (the proxy JID). The MIX channel then broadcasts the presence change to all users who are subscribed to the "urn:xmpp:mix:nodes:presence" node. The presence stanza is sent from the full proxy JID of the client updating status. Note that presence is associated with a client and so will have a full JID. The following example shows a presence message as distributed by the server to a presences subscriber.

Listing 46: Channel Distributes Presence

```
<presence from='123435#coven@mix.shakespeare.example/678'
          to='hag99@shakespeare.example'
          id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'>
  <nick xmlns='http://jabber.org/protocol/nick'>thirdwitch</nick>
  <show>dnd</show>
  <status>Making a Brew</status>
</presence>
```

The presence is distributed to those subscribing to the MIX channel presence node using a standard XMPP presence stanza. The presence change is recorded on the "urn:xmpp:mix:nodes:presence" node. The history of this node will be held as PubSub format in the MAM archive, so that presence history can be viewed.

6.1.8 Client Coming Online and Obtaining Presence from the Local Server

The presence information for a channel is stored in the urn:xmpp:mix:nodes:presence node and distributed using standard presence stanzas to the server of each user subscribing to this presence node. The user's local server will then pass this presence information on to all online clients. This ensures that an online client is kept updated with presence. When a client goes offline, it will cease getting presence updates. Presence updates will continue to flow to the user's local server, and so the local server is able maintain up to date presence state for the channel. The user's server MAY cache this presence information to optimize performance or MAY discard it.
When the client comes online, it will activate use of the MIX. The user's server will then send full presence status to the client using standard presence messages. This will enable the client to update presence information for the channel. Note that this does not need any interaction with the channel.

6.1.9 Updating Presence on User's Server

In normal operation a MIX participant's server will hold accurate presence status for the channel which it provides to clients when they are activated. Incoming presence updates are

immediately sent to active MIX clients.
There are two situations where a MIX participant's server will need to get presence status from the channel. The first time is when a user joins the channel as a participant and subscribes to presence. Upon this subscription the MIX channel will send to the participant's server (using the user's bare JID) presence for all of the items in the presence node using standard presence stanzas. This will give the participant's full current presence for the channel.
The second scenario is when the MIX participant's server needs to load or refresh presence status for a channel. This will be needed when the participant's server is started. This MIX participant's server requests presence update by sending a directed presence stanza to the MIX channel from the user's bare JID. The MIX channel can distinguish this from a presence update, which will always be sent from the clients full JID. This special presence stanza will send to the participant's server (using the user's bare JID) presence for all of the items in the presence node using standard presence stanzas.

6.1.10 Determining Real JIDs

Presence information will provide a MIX client with the nicks and proxy JIDs for participants in a channel. Messages sent to a channel and retrieved from MAM archives will show the proxy JID and nick of the sender. It is sometimes useful to determine the real JID associated with proxy JID. This can always be done for JID Visible channels and can sometimes be done for JID Maybe Visible channels.
For current users JID visible rooms, the real JID is found by a PubSub lookup on the JID Map node. This is shown in the following example:

Listing 47: Client looks up Real JID from Proxy JID

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='kl2fax27'
    to='coven@mix.shakespeare.example'
    type='get'>
     <pubsub xlns='http://jabber.org/protocol/pubsub'>
          <items node='urn:xmpp:mix:nodes:jidmap'>
              <item id='123456#coven@mix.shakespeare.example'/>
          </items>
     </pubsub>
</iq>

<iq from='coven@mix.shakespeare.example'
    id='kl2fax27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
     <pubsub xlns='http://jabber.org/protocol/pubsub'>
          <items node='urn:xmpp:mix:nodes:jidmap'>
             <item id='123456#coven@mix.shakespeare.example'>
                 <participant xmlns='urn:xmpp:mix:1'>
```

```
                    <jid>hecate@mix.shakespeare.example</jid>
                </participant>
            </item>
        </items>
    </pubsub>
  <items>
</iq>
```

For JID Maybe Visible rooms the lookup is performed on the JID Maybe Visible Map node. Note that where a user prefers to not share real JID, the result of this lookup of proxy JID will be the (same) proxy JID.
When an older message is considered, it is possible that the proxy JID of the sender is not current. Such a JID can be looked up in the MAM Archive of the JID Map Node. This is shown in the following example:

Listing 48: Client looks up Real JID from Proxy JID in MAM Archive

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='kl2fax27'
    to='coven@mix.shakespeare.example'
    type='set'>
     <query xlns='urn:xmpp:mam:2'
          queryid='f28'
          node='urn:xmpp:mix:nodes:jidmap'>
            <x xmlns='jabber:x:data' type='submit'>
               <field var='FORM_TYPE' type='hidden'>
                     <value>urn:xmpp:mam:2</value>
               </field>
               <field var='id'>
                    <value>123456#coven@mix.shakespeare.example</value
                       >
                </field>
            </x>
     </query>
</iq>

<message id='iasd208' to='hag66@shakespeare.example/UUID-c8y/1573'>
  <result xmlns='urn:xmpp:mam:2' queryid='f28' id='28482-20987-73623'>
    <forwarded xmlns='urn:xmpp:forward:0'>
      <delay xmlns='urn:xmpp:delay' stamp='2010-07-10T23:08:25Z'/>
         <message xmlns="jabber:client">
                  <event xmlns='http://jabber.org/protocol/pubsub#
                      event'>
                    <items node='urn:xmpp:mix:nodes:jidmap'>
                        <item id='123456#coven@mix.shakespeare.
                            example'>
                            <participant xmlns='urn:xmpp:mix:1'>
                                <jid>hecate@mix.shakespeare.
                                    example</jid>
```

```
                                    </participant>
                                </item>
                            </items>
                        </event>
                </message>
            </forwarded>
        </result>
</message>

<iq from='coven@mix.shakespeare.example'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    id='kl2fax27'
    type='result'>
</iq>
```

6.1.11 Coming Online: Synchronizing Message History

A MIX client will typically display message history of the channel to the user. When a client comes online it will need to obtain this message history from the MAM archive associated with the channel. There are three basic approaches a client will take:

1. If the client has previously displayed message history and has been offline for a reasonably small time, the client MAY wish to retrieve all messages since the last one displayed to the user.
2. The client MAY wish to display a fixed number of messages, perhaps finding more messages if the user subsequently requests.
3. The client MAY wish to display messages from a recent time period, perhaps finding more messages if the user subsequently requests.

To achieve this, the client will query the user's own MAM archive using Message Archive Management (XEP-0313) [32], with the query filtered by the channel JID. This gives the client flexibility to retrieve and display message history in a manner appropriate to the client implementation.
The only exception to this is when a user wishes to access message history in the channel prior to when the user joined the channel. To achieve this, the client will use MAM to retrieve message history directly from the MAM Archive of the MUX channel.

6.1.12 Going Offline

When a client goes offline, this presence update is sent by the client's server to the MIX channel. From the client perspective, this is the same as any other presence change. Handling

[32] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.

by the MIX channel is slightly different.

Listing 49: Client Goes Offline in the Channel

```
<presence type='unavailable'
          from='hag66@shakespeare.example/UUID-a1j/7533'
          to='coven@mix.shakespeare.example'/>
```

The MIX channel will retract (remove) the item in the presence node of the MIX channel identified by the client's full JID. The MIX channel will notify subscribers to the presence node of the user going offline by sending a presence stanza to the full JID of each client. The presence stanza will reference the full proxy JID of the client that is going offline, as shown in the following example:

Listing 50: Channel Distributes Notification of Client going Offline

```
<presence from='12345#coven@mix.shakespeare.example/678'
          to='hecate@shakespeare.example'
          id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
          type='unavailable'/>
```

There is the possibility that the message associated with the user going offline will be lost. If this happens, "ghost" entries will appear in the presence node. A MIX service MAY take steps to address this, for example by probing client with a disco for presence items that remain unchanged for a long period.
It is desirable to prevent clients from going offline briefly and then coming back online again, as this will lead to "flapping presence". The RECOMMENDED approach to achieve this is use of Stream Management (XEP-0198) [33] to maintain an XMPP client connection in the event of short term TCP failure.

6.1.13 Sending a Message

A client sends a message directly to a MIX channel as a standard groupchat message, in exactly the same way as for Multi-User Chat (XEP-0045) [34]. Messages are sent directly to the MIX channel from the user's client. The message id is selected by the client.

Listing 51: User Sends Message to Channel

```
<message from='hag66@shakespeare.example/UUID-a1j/7533'
         to='coven@mix.shakespeare.example'
         id='92vax143g'
         type='groupchat'>
  <body>Harpier cries: 'tis time, 'tis time.</body>
</message>
```

[33] XEP-0198: Stream Management <https://xmpp.org/extensions/xep-0198.html>.
[34] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

The MIX channel then adds information to the message using a <mix> element qualified by the 'urn:xmpp:mix:1' namespace. This element contains two child elements:

1. A <nick> element that contains the Nick of the message sender, taken from the Participants Node.
2. A <jid> element containing the full proxy JID of the sender.

The MIX channel then puts a copy of the message into the MAM archive for the channel and sends a copy of the message to each participant in standard groupchat format. These messages sent by the channel are addressed to the bare JID of each participant and this will be handled by the participant's local server. The message 'from' attribute is the JID of the channel. The id of the message is the ID from the MAM archive and NOT the id used by the sender. The message placed in the MAM archive is the reflected message without a 'to' attribute.

Listing 52: Channel Puts Message in MAM Archive

```
<message from='coven@mix.shakespeare.example'
         id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
         type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
  <mix xmlns='urn:xmpp:mix:1'>
       <nick>thirdwitch</nick>
       <jid>123456#coven@mix.shakespeare.example</jid>
  </mix>
</message>
```

Listing 53: Channel Reflects Message to Participants

```
<message from='coven@mix.shakespeare.example'
         to='hecate@shakespeare.example'
         id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
         type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
  <mix xmlns='urn:xmpp:mix:1'>
       <nick>thirdwitch</nick>
       <jid>123456#coven@mix.shakespeare.example</jid>
  </mix>
</message>
```

The messages sent to participants have a different message id to the originally submitted message. This does not impact most recipients, but it does not allow the message originator to correlate the message with the submitted message. To address this the MIX channel MUST include an additional <submission-id> element in the <mix> element of the message copy going back to the originator's bare JID. The <submission-id> element holds the original id provided by the sender. This enables the originator to correlate the received message with

the message submitted.

Listing 54: Channel Reflects Message back to Originator

```
<message from='coven@mix.shakespeare.example'
         to='hag66@shakespeare.example'
         id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
         type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
 <mix xmlns='urn:xmpp:mix:1'>
        <nick>thirdwitch</nick>
        <jid>123456#coven@mix.shakespeare.example</jid>
        <submission-id>92vax143g</submission-id>
  </mix>
</message>
```

6.1.14 Retracting a Message

A MIX channel MAY support message retraction, where the sender of a messages or an authorized administrator deletes a message. A MIX channel MAY limit the time frame in which a message is allowed to be retracted, for example to prevent retraction of very old messages. When a messages is retracted the original message MAY be replaced by a tombstone. Message retraction is done by sending a special message that identifies the original message. This mechanism allows the retraction to be distributed on the same path as the original message so that all participating servers and clients MAY honour the retraction. The protocol to request retraction does this by adding to a message a <retract> element qualified by the 'urn:xmpp:mix:1' namespace. A retract messages MUST NOT have a body. The <retract> element MUST include an <id> attribute that holds the MAM-ID of the original message. The message sender will need to look up the MAM-ID. The MAM-ID is the convenient message identification for message recipients. A message and it's retraction shown in the following example.

Listing 55: User Retracts a Message

```
<message from='hag66@shakespeare.example/UUID-a1j/7533'
         to='coven@mix.shakespeare.example'
         id='abcde'>
  <body> A Message I did not mean to send </body>
</message>

<message from='hag66@shakespeare.example/UUID-a1j/7533'
         to='coven@mix.shakespeare.example'
         id='92vax143g'>
  <retract id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD' xmlns='
      urn:xmpp:mix:1'/>
</message>
```

The MIX channel will allow a user to retract a message sent by the user if the 'Allow User Message Retraction' option is configured. The MIX channel will allow an administrative user to retract any message if the user is in the group specified by the 'Administrator Message Retraction Rights' option.
If the retraction message is accepted, it MUST be distributed to channel participants. This will allow retraction to happen in the MAM archive of each channel participant and to reflect the retraction in client GUI. A client receiving a retraction message SHOULD ensure that the retracted message is no longer displayed to the end user.
Two approaches to message retraction can be used. In the first approach, the retracted message is simply removed. This is appropriate where retraction is provided as a user service and the user has rights to remove messages sent from the record.
The second approach is to leave a tombstone, which if taken MUST be done in the following manner. It is recommended to use a tombstone, as simply deleting the message may cause confusion with MAM queries. Use of a tombstone is appropriate where it is desired to leave a record of the message that was redacted. With this approach, the original message <body> is removed and replaced with a tombstone using the <retracted> element qualified by the 'urn:xmpp:mix:1' namespace that shows the JID of user performing the retraction and the time of the retraction.

Listing 56: Retracted message tombstone in a MAM result

```
<message id='aeb213' to='juliet@capulet.example/UUID-e3r/9264'>
  <result xmlns='urn:xmpp:mam:1' queryid='f27' id='28482-98726-73623'>
    <forwarded xmlns='urn:xmpp:forward:0'>
      <delay xmlns='urn:xmpp:delay' stamp='2010-07-10T23:08:25Z'/>
      <message xmlns='jabber:client' from="hag66@shakespeare.example"
               to="macbeth@shakespeare.example">
        <retracted xmlns='urn:xmpp:mix:1' by='hag66@shakespeare.
            example'
               time='2010-07-10T23:08:25Z'/>
      </message>
    </forwarded>
  </result>
</message>
```

6.1.15 Telling another User about a Channel

A convenient way to reference another channel is to use References (XEP-0372) [35] which enables the JID of a channel to be shared. This might be used simply to inform the message recipient about the channel or as mechanism to invite the user to join the channel. This is useful as an invitation mechanism to a channel that any user can join or where the invitee knows that the user is allowed to join (e.g., because the channel is for all users in an organization).

[35] XEP-0372: References <https://xmpp.org/extensions/xep-0372.html>.

6.1.16 Inviting another User to join a Channel that the user does not have Permission to Join

Invitation by reference, as described in the previous section, is a convenient approach to invite a user to join a channel that the user has permission to join. This section describes the approach used when the inviter has permission to grant rights for the invitee to become a channel participant. This might be because the inviter is an administrator of the channel or the channel or if the inviter is a channel participant and the channel allows invitation by participants (this channel capability is controlled by the channel configuration variable 'Participation Addition by Invitation from Participant'). Invitation by reference is used to avoid cluttering the allowed node with JIDs of users who are invited to join, but do not accept the invitation. When a channel participant(the inviter) invites another user (the invitee) to join a channel, the following sequence of steps is followed:

1. The inviter checks with disco that the invitee supports MIX.
2. The channel inviter sends to the channel requesting an invite for the invitee.
3. The channel sends an invitation to the inviter.
4. The inviter sends the invitation to the invitee.
5. The invitee MAY use the invitation to join the channel.
6. The invitee MAY send a response to the inviter, indicating if the invitation was accepted or declined.

The first step is for the inviter to request an invitation from the channel. The invitation contains inviter, invitee and a token. The channel will evaluate if the inviter has rights to issue the invitation. This will be because the inviter is a channel administrator or if the inviter is a channel participant and the channel allows invitation by participants. If the inviter has rights to make the invitation, the channel will return a token. The token is a string that the channel can subsequently use to validate an invitation. The format of the token is not specified in this standard. The encoded token MAY reflect a validity time. The invitation request is encoded as an <invite/> child element of an <iq/> element. The <invite/> element is qualified by the 'urn:xmpp:mix:1' namespace. <invite/> contains an <invitation/> child element, which contain <inviter/>, <invitee/>, <channel/> and <token/> child elements.

Listing 57: Inviter Requests and Receives Invitation

```
<iq from='hag66@shakespeare.example/UUID-h5z/0253'
    id='kl2fax27'
    to='coven@mix.shakespeare.example'
    type='get'>
  <invite xmlns='urn:xmpp:mix:1'>
      <invitee>cat@shakespeare.example</invitee>
  </invite>
```

```
</iq>


<iq from='coven@mix.shakespeare.example'
    id='kl2fax27'
    to='hag66@shakespeare.example/UUID-h5z/0253'
    type='result'>
  <invite xmlns='urn:xmpp:mix:1'>
        <invitation>
           <inviter>hag66@shakespeare.example</inviter>
           <invitee>cat@shakespeare.example</invitee>
           <channel>coven@mix.shakespeare.example</channel>
           <token>ABCDEF</token>
        </invitation>
  <invite/>
</iq>
```

The inviter can now send the invitee a message containing the invitation within the <message/> element, as shown in the following example.

Listing 58: Inviter sends Invitation to Invitee

```
<message from='hag66@shakespeare.example/UUID-h5z/0253'
    id='f5pp2toz'
    to='cat@shakespeare.example'>
    <body>Would you like to join the coven?<body>
    <invitation xmlns='urn:xmpp:mix:1'>
        <inviter>hag66@shakespeare.example</inviter>
        <invitee>cat@shakespeare.example</invitee>
        <channel>coven@mix.shakespeare.example</channel>
        <token>ABCDEF</token>
   </invitation>
</iq>
```

The invitation can now be used by the invitee to join a channel. The <invitation/> child element is simply added to the standard channel <join/> element, so that the channel can validate the invitation using the token. If the allowed node is present and the invitee is not matched against any item, the channel MUST add the invitee to the allowed node as part of the join.

Listing 59: User Joins a Channel with an Invitation

```
<iq type='set'
    from='cat@shakespeare.example'
    to='coven@mix.shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
```

```
    <invitation>
        <inviter>hag66@shakespeare.example</inviter>
        <invitee>cat@shakespeare.example</invitee>
        <channel>coven@mix.shakespeare.example</channel>
        <token>ABCDEF</token>
    </invitation>
  </join>
</iq>
```

The invitee MAY send an acknowledgement back to the inviter, noting the status of the invitation. This is encoded as an <invitation-ack/> child element of <message/> element. The <invitation-ack/> element is qualified by the 'urn:xmpp:mix:1' namespace. The <invitation-ack/> has an <invitation/> child element that encodes the invitation being acknowledged and a <value/> child element to encode the acknowledgement value. <value/> has the following values:

- 'Joined': The invitee has joined the channel.
- 'Declined': The invitee is not taking up the invitation.
- 'Acknowledged': The invitation is acknowledged, without information on action taken or planned.

Listing 60: Invitee sends Acknowledgement to Inviter

```
<message from='cat@shakespeare.example/UUID-l1w/8813'
    id='b6p9llze'
    to='hag66@shakespeare.example/UUID-h5z/0253'>
    <body>No Thanks - too busy chasing mice....<body>
    <invitation-ack xmlns='urn:xmpp:mix:1'>
        <value>Declined</value>
        <invitation>
            <inviter>hag66@shakespeare.example</inviter>
            <invitee>cat@shakespeare.example</invitee>
             <channel>coven@mix.shakespeare.example</channel>
             <token>ABCDEF</token>
         </invitation>
     </invitation-ack>
</iq>
```

6.1.17 Sending Private Messages

Private Messages are used to provide communication with another channel participant through the MIX channel, without the initiating user knowing the real JID of the channel participant. A channel MAY support use of Private Messages. Private messages are standard XMPP messages and MUST NOT be groupchat. A message goes through a number of stages with different addressing. This is set out in the following table.

Message	From	To
First Message from Client to MIX Channel	Full JID of initiator's client	Proxy bare JID of responder
First Message from MIX Channel to responder's server	Proxy full JID of initiator's client	Bare JID of responder
First Message from responder's server to one or more of the responder's clients	Proxy full JID of initiator	Full JID of responder's client
Messages from responder's client to MIX Channel	Full JID of responder's client	Proxy full JID of initiator's client
Messages from MIX channel to initiator's client	Proxy full JID of responder's client	Full JID of initiator's client

Message	From	To
Messages from initiator's client to MIX Channel	Full JID of initiator's client	Proxy full JID of responder's client
Message from MIX Channel to responder's client	Proxy full JID of initiator's client	Full JID of responder's client

Private Messages MAY be archived using MAM by the XMPP servers associated with initiator and responder. Private Messages MUST NOT be archived by the MIX channel.

6.1.18 Requesting vCard

A client MAY request the vCard of a channel participant by sending a request through the channel. The MIX channel MAY pass this request on or MAY block it. vCard requests MAY use vcard-temp (XEP-0054) [36] (vcard-temp) or vCard4 over XMPP (XEP-0292) [37] (vCard4 over XMPP). The MIX channel does not process the vCard requests, but simply relays them on to real bare JID of the target. A MIX service MAY choose to relay one or both protocols. Where a MIX service relays one or both of these protocols, each protocol relayed MUST be advertised as a feature of the MIX service. In the following example, using vcard-temp, the requesting client sends a message to the bare proxy JID of the channel participant for which the vCard is desired.

Listing 61: Client directly requests vCard through channel

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='lx09df27'
    to='989898#coven@mix.shakespeare.example'
    type='get'>
  <vCard xmlns='vcard-temp'/>
</iq>
```

[36] XEP-0054: vcard-temp <https://xmpp.org/extensions/xep-0054.html>.
[37] XEP-0292: vCard4 over XMPP <https://xmpp.org/extensions/xep-0292.html>.

The MIX channel MAY pass on the vCard request or MAY reject with an error, dependent on channel policy. The MIX service will then address the vCard request to the user's server (using bare JID) using a full proxy JID to hide the requester.

Listing 62: Channel passes on vCard request to the User's Server

```
<iq from='123456#coven@mix.shakespeare.example/6789'
    id='lx09df27'
    to='peter@shakespeare.example'
    type='get'>
  <vCard xmlns='vcard-temp'/>
</iq>
```

The user's server, on behalf of the user, MAY send a response or reject with an error. The user's server will send the vCard back to the channel.

Listing 63: User's Server sends vCard Response via MIX channel

```
<iq from='peter@shakespeare.example'
    id='lx09df27'
    to='123456#coven@mix.shakespeare.example/6789'
    type='result'>
    <vCard xmlns='vcard-temp'>
    <FN>Peter Saint-Andre</FN>
    <N>
      <FAMILY>Saint-Andre</FAMILY>
      <GIVEN>Peter</GIVEN>
      <MIDDLE/>
    </N>
    <NICKNAME>stpeter</NICKNAME>
    <URL>http://www.xmpp.org/xsf/people/stpeter.shtml</URL>
  </vCard>
  <query xmlns='http://jabber.org/protocol/disco#info'>
</iq>
```

The MIX channel will then send the vCard response to the requesting client on behalf of the client sending the response.

Listing 64: MIX Channel sends vCard responst to Client

```
<iq from='989898#coven@mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
 <vCard xmlns='vcard-temp'>
    <FN>Peter Saint-Andre</FN>
    <N>
      <FAMILY>Saint-Andre</FAMILY>
```

```
      <GIVEN>Peter</GIVEN>
      <MIDDLE/>
    </N>
    <NICKNAME>stpeter</NICKNAME>
    <URL>http://www.xmpp.org/xsf/people/stpeter.shtml</URL>
  </vCard>
</iq>
```

6.2 Presence Initializion

For an active MIX Channel, the presence node is updated as channel participants change status and presence information is sent to the channel. When a MIX channel starts, typically when the associated MIX Service and Server start, there is a need to initialize the presence node. This is done by the XMPP server associated with the MIX channel sending out a presence probe for each channel participant, following the presence probe process specified in RFC 6121 [38]. A presence probe MUST NOT be sent for users who have set presence preference to not sharing.

6.3 Ensuring Message Delivery

It is important that messages are all transferred from the MIX channel to the server associated with the each channel participant. Transfer between servers will typically happen quickly and Stream Management (XEP-0198) [39] will deal with short connection failures between servers. Longer connection failures could lead to message loss. This would lead to online users (who remain connected to their servers) missing messages, and to messages being missed out of the user archive. This section describes how MIX addresses this.
When there is a long term connection failure, the MIX channel will receive an error from the XMPP server indicating that a message failed to transfer to a recipient. When this happens, the MIX channel MUST take responsibility to ensure that the message is retransmitted and delivered. When the MIX channel detects a failure it will make use of an IQ Marker message to determine when the connection to the peer server is working again. Once the channel has received a response to the marker IQ it will retransmit the pending messages. The marker is encoded as a <marker/> child element of an <iq/> element. The <marker/> element is qualified by the 'urn:xmpp:mix:1' namespace.

Listing 65: Channel Sends Marker Message

```
<iq from='coven@mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example'
    type='get'>
  <marker xmlns='urn:xmpp:mix:1'/>
```

[38] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[39] XEP-0198: Stream Management <https://xmpp.org/extensions/xep-0198.html>.

```
</iq>


<iq from='hag66@shakespeare.example'
    id='lx09df27'
    to='coven@mix.shakespeare.example'
    type='result'>
 <marker xmlns='urn:xmpp:mix:1'/>
</iq>
```

6.4 Use of MAM

All messages sent to a MIX channel MUST be archived using MAM in association with the MIX channel. All messages MUST also be archived on the server associated with each channel participant receiving the message, which enables clients to always retrieve messages from the clients MAM archive. Other channel nodes MAY be archived.

6.4.1 Archive of Messages

Messages sent to participants MUST be archived by both the MIX channel and by the user's server. This MAY include presence messages. Correct MIX operation relies on messages being archived.

6.4.2 Retrieving Messages

Clients will retrieve MIX messages using standard MAM protocol from the user's archive. The MAM query will filter based on the channel JID to enable access to messages from a given channel. This gives the user a simple mechanism to access all messages sent to the channel. MAM can be used to retrieve older messages that have not been cached by the client.
Messages can also be retrieved from the channel by addressing MAM queries to the channel JID. This will behave like a standard MAM archive. This can be useful for administrators to access archived messages. It can also be useful for new channel participants to access the historical archives.

6.4.3 MAM Use with other Channel Nodes

A MIX Channel MAY use MAM to archive nodes other than message nodes. Clients with rights to access these archives MAY use MAM to do this, specifying the PubSub node in the MAM query addressed to the channel.

6.5 Administrative Use Cases

6.5.1 Checking For Permission To Create a Channel

MIX does not standardize an access control model for creating and deleting MIX channels. The choice is left to the MIX implementer, and could be a very simple or complex approach. A client can determine if it has permission to create a channel on a MIX service, which MAY be used to control options presented to the user. This is achieved by a disco command on the MIX service. If the 'urn:xmpp:mix:1#create-channel' feature is returned, the user is able to create a channel.

Listing 66: Client determines Capability to Create a Channel

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='Shakespearean Chat Service'
        type='text'/>
    <feature var='urn:xmpp:mix:1'/>
    <feature var='urn:xmpp:mix:1#create-channel'>
  </query>
</iq>
```

6.5.2 Creating a Channel

A client creates a channel by sending a simple request to the MIX service. A channel MAY be created with default parameters, as shown in the following example. The result MUST include the name of the channel which MUST match the channel name in the request (if present). The create is encoded as a <create/> child element of <iq/> element. The <create/> is qualified by the 'urn:xmpp:mix:1' namespace. The <create/> element MUST have a 'channel' attribute to specify the channel name. This attribute specifies the value that will be used in the LHS of the JID for the MIX channel.

Listing 67: Creating a Channel with Default Parameters

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
```

```
    id='lx09df27'
    to='mix.shakespeare.example'
    type='set'>
  <create channel='coven' xmlns='urn:xmpp:mix:1'/>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
  <create channel='coven' xmlns='urn:xmpp:mix:1'/>
</iq>
```

The client MAY also include parameters in Data Forms (XEP-0004) [40] format as part of channel creation. If the client wishes to inspect configuration, channel administration procedures SHOULD be used.

Listing 68: Creating a Channel with Client Specified Parameters

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='set'>
  <create channel='coven' xmlns='urn:xmpp:mix:1'>
     <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <field var='Owner'>
            <value>hecate@shakespeare.example</value>
            <value>greymalkin@shakespeare.example</value>
        </field>
        <field var='Messages␣Node␣Subscription'>
           <value>allowed</value>
         </field>
        <field var='JID␣Visibility'>
           <value>jid-mandatory-visible</value>
         </field>
         <field var='No␣Private␣Messages'>
            <value>true</value>
         </field>
         </x>
  </create>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
```

[40] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

```
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
  <create channel='coven' xmlns='urn:xmpp:mix:1'/>
</iq>
```

6.5.3 Creating a Channel for Ad Hoc Use

Channels MAY be created for ad hoc use between a set of users. Channels of this nature will have channel name created by the server and will not be addressable or discoverable. Here a channel is created without specifying the channel name. Parameters for the channel MAY also be specified.

Listing 69: Creating a Channel for Ad Hoc Use

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='set'>
  <create xmlns='urn:xmpp:mix:1'/>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
  <create channel='A1B2C345' xmlns='urn:xmpp:mix:1'/>
</iq>
```

6.5.4 Converting a 1:1 Conversation to a Channel

A common use case for an ad hoc channel is where two users are engaged in a 1:1 chat and wish to broaden the discussion. Prior to bringing more users into a channel, using standard invitation process, there is a need to create and populate a channel. The first step is for one of the two users to create an ad hoc channel, as described in the previous section. The other user will then be invited, and can switch to the new channel.
It can also be useful to share some or all of the messages from the 1:1 discussion into the new channel. The mechanism to do this is to forward messages to be shared in the MUC using Stanza Forwarding (XEP-0297) [41]. A body SHOULD NOT be used in the outer message. Sharing history is optional. If history is shared, it MUST be done by the user creating the channel before the other user is invited. Any other use of forwarded messages MUST be treated as a member of the MUC forwarding a message to the channel and MUST NOT be treated as history sharing.

[41] XEP-0297: Stanza Forwarding <https://xmpp.org/extensions/xep-0297.html>.

Listing 70: Forwarding a message to create History

```
<message from='hag66@shakespeare.example/UUID-a1j/7533'
         to='A1B2C345@mix.shakespeare.example'
         id='92vax143g'
         type='groupchat'>
         <forwarded xmlns='urn:xmpp:forward:0'>
          <delay xmlns='urn:xmpp:delay' stamp='2010-07-10T23:08:25Z'/>
          <message from='hag67@shakespeare.example/pda'
                   id='0202197'
                   to='hag66@shakespeare.example/UUID-a1j/7533'
                   type='chat'
                   xmlns='jabber:client'>
              <body>Harpier cries: 'tis␣time,␣'tis time.</body>
          </message>
        </forwarded>
</message>
```

There are a number of security considerations with use of MUC history. There may be sensitive information in the 1:1 MUC history, and the user sharing this history should ensure that none of this is sensitive, prior to sharing in this way. The user creating the MUC has potential to inject history messages which were not part of the history. It is recommended that the second user joining the MUC to validate that the messages match the history and to take appropriate action if they do not.

6.5.5 Destroying a Channel

MIX channels are always explicitly destroyed by an owner of the channel or by a server operator. There is no concept of temporary channel, equivalent to Multi-User Chat (XEP-0045) [42] temporary room which is automatically destroyed by the server when the users leave. However, channels MAY be configured with an explicit lifetime, after which the channel MUST be removed by the MIX service. Where a channel is created for ad hoc use, it MAY be desirable to keep the channel for history reference or for re-use by the same set of users. Note that the owner of the channel does not need to have presence registered in the channel in order to destroy it.
The destroy operation is encoded as a <destroy/> child element of an <iq/> element. The <destroy/> element is qualified by the 'urn:xmpp:mix:1' namespace. The <destroy/> element MUST have a 'channel' attribute to specify the channel to be destroyed. A client destroys a channel using a simple set operation, as shown in the following example.

Listing 71: Client Destroys a Channel

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
```

[42] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

```
    type='set'>
  <destroy channel='coven' xmlns='urn:xmpp:mix:1'/>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
</iq>
```

6.5.6 Server Destroying a Channel

A server MUST destroy a channel that has exceeded its specified explicit lifetime. Servers MAY destroy channels which have no participants and/or presence according to local policy. There will often be good reasons to not destroy rooms in these scenarios, in particular to facilitate channel re-use and history access.

6.5.7 Modifying Channel Information

Authorized users, typically owners and sometimes administrators, MAY modify the channel information. The client MAY issue a pubsub get command to obtain a form that will facilitate update of the information node. The values in the form show current values, which be defaults or MAY have been explicitly set. In the following example, the channel name was previously set, but other values were not.

Listing 72: Getting Information Form

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='get'>
     <pubsub xmlns='http://jabber.org/protocol/'>
         <items node='urn:xmpp:mix:nodes:info'/>
     </pubsub>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
    <pubsub xmlns='http://jabber.org/protocol/'>
     <items node='urn:xmpp:mix:nodes:info'>
       <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
```

```
        <title>Information Node Modification</title>
        <field type='text-multi'
               label='Channel Name'
               var='Name'>
            <value>Witches Coven</value>
        </field>
        <field type='text-multi'
               label='Channel Description'
               var='Description'/>
         <field type=jid-single'
                label='Channel Administrative Contact'
                var='Contact'/>
      </x>
    </items>
  </pubsub>
</iq>
```

Updating the information node is done using a pubsub set command. The MIX channel MUST update the fields with values provided, leaving other fields unchanged. The result returns the id used in the information node item, which is the date/time of the modification.

Listing 73: Modifying Channel Information

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='set'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
     <publish node='urn:xmpp:mix:nodes:info'>
       <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <field var='Name'>
            <value>Witches Coven</value>
        </field>
        <field var='Description'>
           <value>A location not far from the blasted heath where
                        the three witches meet</value>
         </field>
         <field var='Contact'>
           <value>greymalkin@shakespeare.example</value>
         </field>
      </x>
     </publish>
  </pubsub>
</iq>

<iq from='mix.shakespeare.example'
```

```
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <publish node='urn:xmpp:mix:nodes:info'>
        <item id='2016-05-30T09:00:00' xmlns='urn:xmpp:mix:1'/>
       </publish>
   </pubsub>
</iq>
```

6.5.8 Modifying Channel Configuration

Channel owners are allowed to modify the channel configuration. The client MAY issue a pubsub get command to obtain a form that will facilitate update of the configuration node. Other clients MAY be authorized to use this command to see the channel configuration, but only owners MAY update the configuration. The values in the form show current values, which MAY be defaults or MAY have been explicitly set. The following example shows a short form returned to illustrate the syntax. A typical configuration form will be much larger with many fields. Modifying channel configuration is done directly by a client.

Listing 74: Getting Configuration Form

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='get'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <items node='urn:xmpp:mix:nodes:config'/>
     </pubsub>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
     <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <items xmlns='urn:xmpp:mix:1'  node='urn:xmpp:mix:nodes:config'>
       <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <title>Configuration Node Modification</title>
         <field type=jid-multi'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣label='Channel Administrator'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣var='Administrator'/>
␣␣␣␣␣␣</x>
␣␣␣␣␣</items>
```

```
␣␣␣</pubsub>
</iq>
```

Updating the information node is done using a pubsub set command. The MIX channel MUST update the fields with values provided, leaving other fields unchanged. The result returns the id used in the configuration node item, which is the date/time of the modification.

Listing 75: Modifying Channel Configuration

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='set'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <publish node='urn:xmpp:mix:nodes:config'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
             <value>urn:xmpp:mix:1</value>
        </field>
        <field var='Owner'>
            <value>hecate@shakespeare.example</value>
            <value>greymalkin@shakespeare.example</value>
        </field>
        <field var='Messages␣Node␣Subscription'>
           <value>allowed</value>
         </field>
        <field var='JID␣Visibility'>
           <value>jid-mandatory-visible</value>
         </field>
         <field var='No␣Private␣Messages'>
            <value>true</value>
         </field>
      </x>
     </publish>
   </pubsub>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <publish node='urn:xmpp:mix:nodes:config'>
        <item id='2016-05-30T09:00:00' xmlns='urn:xmpp:mix:1'/>
       </publish>
   </pubsub>
</iq>
```

6.5.9 Controlling Channel Partipcipants

Owners and Administrators are allowed to control which users can participate in a channel by use of Allowed and Banned lists using PubSub. These operations follow Publish-Subscribe (XEP-0060) [43] which sets out detailed protocol use and error handling. Allowed and Banned lists MAY be read by PubSub get of the Banned and Allowed Nodes. This operation MAY be used by users as controlled by 'Allowed Node Subscription' and 'Banned Node Subscription' configuration node options (default Administrators).

Listing 76: Client Reads Allowed Node

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='get'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
         <items node='urn:xmpp:mix:nodes:allowed'/>
    </pubsub>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
         <items node='urn:xmpp:mix:nodes:allowed'>
            <item id='shakespeare.example'/>
            <item id='alice@wonderland.example'/>
         </items>
    </pubsub>
</iq>
```

JIDs can be added to the Allowed and Banned nodes by a pubsub set command. This is used to add one item to a node.

Listing 77: Client Adds a JID to the Allowed Node

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='set'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
        <publish node='urn:xmpp:mix:nodes:allowed'>
            <item id='marlow.example'/>
         </items>
    </pubsub>
</iq>
```

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

```
<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'/>
</iq>
```

JIDs can be removed from the Allowed and Banned nodes by pubsub retract command.

Listing 78: Client Removes a JID to the Banned Node

```
<iq from='hag66@shakespeare.example/UUID-a1j/7533'
    id='lx09df27'
    to='mix.shakespeare.example'
    type='set'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
        <retract node='urn:xmpp:mix:nodes:banned'>
            <item id='lear@shakespeare.example'/>
         </items>
    </pubsub>
</iq>

<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-a1j/7533'
    type='result'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'/>
</iq>
```

When the MIX channel adds a JID to the banned node, other nodes in the MIX channel will be appropriately updated to reflect this change. In particular, the participants nodes and presence nodes will be updated to remove matching JIDs. This will have the effect of immediately removing the user from the channel. For this reason, there is no requirement to have the "kick" functionality of MUC, as this is achieved by banning the user.

7 MIX Requirements on Participant's Server

This section defines behaviour REQUIRED by MIX for servers supporting MIX participants. This provides the full MIX specification for clients and servers is set out in a single document. This functionality MUST be provided by servers used by clients that participate in MIX channels. In future, the specifications in this section MAY be moved to a separate XEP or it MAY be incorporated into Pubsub Account Management (XEP-0376) [44] (PAM) which follows a similar model.

[44] XEP-0376: Pubsub Account Management <https://xmpp.org/extensions/xep-0376.html>.

7.1 Server Identification of MIX Capabable Clients

A MIX User's server MUST determine which online clients support MIX. This will enable the server to send MIX traffic to all MIX capable clients. A MIX capable client MAY choose to come online and not advertise MIX capability. The mechanism for a server to discover client capability is described in Discovering Client MIX Capability.

7.2 Messages From MIX Channels

Messages from a MIX channel will usually go to the participant's server. The only exception to this is where the MIX channel is responding directly to messages from the client. Messages and presence distributed but a MIX channel will always be sent to the participant's server and addressed to the user's bare JID. The participant's server will simply send on the messages from the channel to each of the user's online clients which advertise MIX capability. If there are no such clients activated, the message is dropped.
Messages sent to the participant's sever will always be addressed to the user's bare JID. The participant's server will modify the recipient to the full JID of each client to which the message is forwarded. The participant's server MUST NOT make any other modifications to each message. The following example, repeated from earlier, shows a message distributed by a MIX channel and directed to the participant's server with the participant's bare JID.

Listing 79: Channel Reflects Message to Participants

```
<message from='coven@mix.shakespeare.example'
         to='hecate@shakespeare.example'
         id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
         type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
  <mix xmlns='urn:xmpp:mix:1'>
       <nick>thirdwitch</nick>
       <jid>123456#coven@mix.shakespeare.example</jid>
  </mix>
</message>
```

The server receiving the message will then deliver the messages to all online clients. Messages are delivered to all available online resources irrespective of status and resource priority. The following example shows how the participant's server modifies the inbound message to replace the bare JID in the 'to' with a full JID for each of two active MIX clients.

Listing 80: Participant's Server Sends Modified Message to two Clients

```
<message from='coven@mix.shakespeare.example'
         to='hecate@shakespeare.example/UUID-x4r/2491'
         id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
         type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
```

```
  <mix xmlns='urn:xmpp:mix:1'>
        <nick>thirdwitch</nick>
        <jid>123456#coven@mix.shakespeare.example</jid>
  </mix>
</message>

<message from='coven@mix.shakespeare.example'
         to='hecate@shakespeare.example/UUID-b5b/0114'
         id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
         type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
   <mix xmlns='urn:xmpp:mix:1'>
        <nick>thirdwitch</nick>
        <jid>123456#coven@mix.shakespeare.example</jid>
  </mix>
</message>
```

7.3 Messages To MIX Channels

Messages sent by a MIX channel participant to the MIX channel are always sent from a MIX client directly to the channel. The participant's server relays the message but does not modify the messages.

7.4 Client Determines MIX Capability of Local Server

Servers supporting the capabilities necessary to enable MIX clients MUST advertise this. A client wishing to use MIX MUST check for this capability in the server before using MIX. The capability is represented by the 'urn:xmpp:mix:account:0' feature.

Listing 81: Client Determines MIX Capability of Local Server

```
<iq from='hag66@shakespeare.example/UUID-c8y/1573'
    id='lx09df27'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq from='shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='urn:xmpp:mix:account:0'/>
  </query>
</iq>
```

7.5 MIX Management and Discovery

Most interaction between a MIX client and a MIX channel is directly between the client and the channel. The participant's server relays the message but does not modify the messages. In particular configuration management and discovery is direct. Interaction will be direct, unless explicitly stated otherwise.

7.6 IQ Support on Local Server

Channel Join and Leave functions operate indirectly through the participant's server. The reason for this is that where a channel shares user presence, the channel is included in the user's roster which is managed in the local server. The Join and Leave functions lead to roster changes and so they MUST go through the participant's server. This is included in each of the operations that work in this manner. The general mechanism to achieve this is illustrated by example in this section. The client addresses indirect messages to the user's own bare JID, indicating the channel with the channel attribute. This is illustrated below.

Listing 82: Client sends request to local server to Join a MIX Channel

```
<iq type='set'
    from='hag66@shakespeare.example/UUID-a1j/7533'
    to='hag66@shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1'
        channel='coven@mix.shakespeare.example'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <subscribe node='urn:xmpp:mix:nodes:presence'/>
    <subscribe node='urn:xmpp:mix:nodes:participants'/>
    <subscribe node='urn:xmpp:mix:nodes:config'/>
  </join>
</iq>
```

This is then modified by the local server and sent to the MIX channel. This is shown in the following example, repeated from the earlier specification of channel behaviour.

Listing 83: Participant's Server sends Join to MIX Channel

```
<iq type='set'
    from='hag66@shakespeare.example'
    to='coven@mix.shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <join xmlns='urn:xmpp:mix:1'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <subscribe node='urn:xmpp:mix:nodes:presence'/>
    <subscribe node='urn:xmpp:mix:nodes:participants'/>
    <subscribe node='urn:xmpp:mix:nodes:config'/>
```

```
  </join>
</iq>
```

The MIX service will send the IQ response to indirect messages to the user's server using the user's bare JID. The user's server will then route the response back to the user's client.

7.7 Roster Management

The participant's server is responsible for ensuring that MIX channels are correctly entered into the user's roster when user's join MIX channels and for removing them when they leave. The participant's server MUST ensure that only presence information from clients that advertise MIX capability is sent to the MIX channel. So, if a user has two online clients, but only one is MIX capable, then the channel will only receive presence information relating to the MIX client.

7.8 MIX Roster Item Capability Sharing

It is useful for a MIX client to know which roster members are MIX channels, as this will facilitate convenient presentation of subscribed MIX channels to the user. A MIX client MAY request that the server return this additional information that annotates roster elements with MIX capability. The server MUST return the additional information. The request is made by extending the standard roster get request by adding a child element <annotate/> to the <query/> element. The <annotate/> element is qualified by the 'urn:xmpp:mix:roster:0' namespace.

Listing 84: Roster Get Requesting MIX Information

```
<iq from='juliet@example.com/balcony'
          id='bv1bs71f'
          type='get'>
       <query xmlns='jabber:iq:roster'>
              <annotate xmlns‘=urn:xmpp:mix:roster:0'/>
␣␣␣␣␣␣␣<query/>
␣␣␣␣␣</iq>
```

A standard roster item is encoded as follows.

Listing 85: Standard Roster Item Encoding

```
<item jid='romeo@example.net'/>
```

MIX channels in the roster information returned in response to a request for this additional MIX information MUST have an element <channel/> qualified by the 'urn:xmpp:mix:roster:0' namespace included in the roster item, as shown inf the following example.

Listing 86: Roster Item Encoding of a MIX Channel

```
<item jid='balcony@example.net'>
      <channel xmlns‘=urn:xmpp:mix:roster:0'/>
  </item>
```

7.9 MAM Archive Support

Archive of MIX channel messages is done by the participant's server. Where a message is sent to the participant's server and discarded because there are no active clients, it will still be archived. This means that the messages will be available in the local archive and can be picked up by clients when they come online.

8 Supporting MIX and MUC together

MIX is specified as a service that can be used independent of MUC and a MIX service MAY be implemented without MUC. If both MIX and MUC are implemented, three approaches are noted.

1. Entirely separate MIX and MUC implementation, with MIX and MUC using separate domains and MIX channels being completely separate from MUC rooms.
2. Fully integrated MIX and MUC implementation, with MIX and MUC sharing a single domain and rooms/channels equivalent.
3. Partially integrated MIX and MUC implementation, with MIX and MUC using separate domains, but rooms/channel are common. This means that each MIX channel will have MUC room of the same name and same participants.

The fully integrated approach would be transparent to clients. The following example shows how a MIX service that also supported MUC would respond:

Listing 87: Service responds with Disco Info result showing MIX and MUC support

```
<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='Shakespearean Chat Service'
        type='text'/>
    <feature var='urn:xmpp:mix:1'/>
    <feature var='http://jabber.org/protocol/muc'/>
```

```
    <x xmlns='jabber:x:data' type='result'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:mix:1#serviceinfo</value>
      </field>
    </x>
  </query>
</iq>
```

In the fully integrated service item discovery on the MIX/MUC service determines a list of channels. The protocol used for this is the same in MUC and MIX. Discovery actions on a channel in MIX MUST use 'node=mix' attribute in the discovery which will lead to the return of MIX channel specific information, as mandated for this discovery in MIX. If is not set, MUC room specific information is returned.
For the partially integrated service, it will be useful for clients that support both MIX and MUC to be able to determine that the server supports both protocols. For a MIX client, it will be useful to know the MUC service, so that this information can be shared with a MUC client invitation. This information is provided by the initial service discovery:

Listing 88: MIX Service responds with Disco Info result sharing MUC service location

```
<iq from='mix.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='Shakespearean Chat Service'
        type='text'/>
    <feature var='urn:xmpp:mix:1'/>
    <x xmlns='jabber:x:data' type='result'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:mix:1#serviceinfo</value>
      </field>
      <field var='muc-mirror'
             type='jid-single'
             label='Location of MUC mirror service'>
        <value>chat.shakespeare.example</value>
      </field>
    </x>
  </query>
</iq>
```

The result is returned in an extended disco results in a form whose type value is 'urn:xmpp:mix:1#serviceinfo'. The field with var='muc-mirror' is the value of which is the mirrored MUC domain's JID.
Where a client supporting both MIX and MUC is given a reference to a MUC room, it is desirable that the client can determine the MIX channel and join using MIX. This is achieved

by an equivalent extension to MUC service discover.

Listing 89: MUC Service responds with Disco Info result sharing MIX service location

```
<iq from='chat.shakespeare.example'
    id='lx09df27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='Shakespearean Chat Service'
        type='text'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <x xmlns='jabber:x:data' type='result'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:mix:1#serviceinfo</value>
      </field>
      <field var='mix-mirror'
             type='jid-single'
             label='Location of MUC mirror service'>
        <value>mix.shakespeare.example</value>
      </field>
    </x>
  </query>
</iq>
```

The result is returned in an extended disco results in a form whose type value is 'urn:xmpp:mix:1#serviceinfo'. The field with var='mix-mirror' is the value of which is the mirrored MIX domain's JID.

8.1 Choosing Which Invite to Send

Where a client supports MUC and MIX and has determined that for a channel that the server also supports a MUC room, the client has a choice as to which type of invite to send. This SHOULD be done by determining if the client support MIX using the mechanism specified in Discovering Client MIX Capability. If the client supports MIX a MIX invite SHOULD be sent.

9 Capabilities not provided in MIX

This section lists a number of capabilities not specified in the core MIX which are provided in Multi-User Chat (XEP-0045) [45]. These capabilities will not be added to core MIX but they could

[45] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

in the future be specified as independent XEPs.

9.1 Password Controlled Channels

Multi-User Chat (XEP-0045) [46] provides a mechanism to control access to MUC rooms using passwords. An equivalent mechanism is not included in core MIX, as it has a number of security issues. Control of access to channels is better achieved using an explicit list of participants.

9.2 Voice Control

Multi-User Chat (XEP-0045) [47] defines a mechanism so that MUC moderators can control who is able to send messages to a MUC room using a ”voice” mechanism. MIX does not provide an exact functional equivalent, although access control to channels enables some of the goals of voice control to be achieved in a different manner.

9.3 Subject

Multi-User Chat (XEP-0045) [48] provide a Subject capability to enable setting of the current topic of discussion. The Name and Description attributes provided by MIX enable descriptive information to be associated with a channel. These attributes can replace Subject in the way it is used in many MUC rooms, but they do not reflect the more limited topic nature of Subject. It is likely that a new XEP to be used with MIX will be written, perhaps using a ”Sticky Messages” approach to fulfil the Subject capability using a different approach.

10 Internationalization Considerations

MIX allows specification of a number of human readable strings associated with a MIX channel, in particular the name and description information of a MIX channel. These strings MAY have language set using an xml:lang attribute, and multiple values MAY be set provided that each one is distinguished using xml:lang.
Nicknames SHOULD be normalized using the ”nickname” profile of the PRECIS OpaqueString class, as defined in RFC 7700 [49].

[46] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[47] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[48] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[49] RFC 7700: Preparation, Enforcement, and Comparison of Internationalized Strings Representing Nicknames<http://tools.ietf.org/html/rfc7700>.

11 Security Considerations

MIX is built over MAM and PubSub and the security considerations of Message Archive Management (XEP-0313) [50] and Publish-Subscribe (XEP-0060) [51] MUST be considered. These services protect MIX channel information, which can be sensitive and needs appropriate protection.
MIX channels MAY be JID Hidden, in order to hide the JIDs of channel participants from those accessing the channel. Care MUST be taken to ensure that JIDs are fully hidden. In particular when proxy JIDs are prepared, this MUST be done in a manner which ensure that the real JIDs cannot be determined. Where nicks are assigned by a channel, this MUST be done in a way that does not expose the JID.
There is no MIX equivalent to Multi-User Chat (XEP-0045) [52] password controlled rooms, which avoids a number of security issues.
MIX provides flexible access control options, which MUST be used in a manner appropriate to the security requirements of MIX users and services.
When converting a 1:1 conversation to a channel there is potential to expose sensitive information and to present invalid information.

12 IANA Considerations

None.

13 XMPP Registrar Considerations

The urn:xmpp:mix namespace needs to be registered.

14 XML Schema

To be supplied when MIX progresses to proposed standard.

15 Acknowledgements

Thanks to the following who have made contributions: Dave Cridland, Tarun Gupta, Philipp Hancke, Waqas Hussain, Timothée Jaussoin, Evgeny Khramtsov, Georg Lukas, Tobias Markmann, Ralph Meijer, Edwin Mons, Emmanuel Gil Peyrot, Florian Schmaus, Lance Stout, Sam

[50] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
[51] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[52] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

Whited, Jonas Wielicki, Matthew Wild and one anonymous reviewer.