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

2020-12-01
Version 0.14.6

Status	Type	Short Name
Experimental	Standards Track	MIX-CORE

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.

MIX 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.
15. MIX should be suitable for use by humans and as a building block for other clients.

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

3 MIX Specification Family

MIX is specified as a family of XEPs that address the full set of requirements. Only two of these XEPs are mandatory for providing a MIX service. The others provide additional services and are used when these additional services are required. Each of these specifications has a short name, which is used to refer the specific XEP. The term MIX is used to refer to the family of XEPs. MIX is extensible, and it is anticipated that further XEPs will be added to this family. The XEPs are:

1. MIX-CORE. Mediated Information eXchange (MIX) (XEP-0369) [6]. This specification is the central mandatory MIX specification. It sets out requirements addressed by MIX and general MIX concepts and framework. It defines joining channels and associated participant management. It defines discovery and sharing of MIX channels and information about them. It defines use of MIX to share messages with channel participants.

2. MIX-PRESENCE. Mediated Information eXchange (MIX): Presence Support. (XEP-0403) [7]. This optional specification adds the ability for MIX online clients to share presence, so that this can be seen by other MIX clients. It also specifies relay of IQ stanzas through a channel.

3. MIX-PAM. Mediated Information eXchange (MIX): Participant Server Requirements (XEP-0405) [8]. This specification defines how a server supporting MIX clients behaves, to support servers implementing MIX-CORE and MIX-PRESENCE.

4. MIX-ADMIN. Mediated Information eXchange (MIX): MIX Administration (XEP-0406) [9]. This specifies MIX configuration and administration of MIX.

5. MIX-ANON. Mediated Information eXchange (MIX): JID Hidden Channels. (XEP-0404) [10]. This specifies a mechanism to hide real JIDs from MIX clients and related privacy controls. It also specifies private messages.

6. MIX-MISC. Mediated Information eXchange (MIX): Miscellaneous Capabilities (XEP-0407) [11]. This specifies a number of small MIX capabilities which are useful but do not need to be a part of MIX-CORE.

[6] XEP-0369: Mediated Information eXchange (MIX) <https://xmpp.org/extensions/xep-0369.html>.
[7] XEP-0403: Mediated Information eXchange (MIX): Presence Support. <https://xmpp.org/extensions/xep-0403.html>.
[8] XEP-0405: Mediated Information eXchange (MIX): Participant Server Requirements <https://xmpp.org/extensions/xep-0405.html>.
[9] XEP-0406: Mediated Information eXchange (MIX): MIX Administration <https://xmpp.org/extensions/xep-0406.html>.
[10] XEP-0404: Mediated Information eXchange (MIX): JID Hidden Channels. <https://xmpp.org/extensions/xep-0404.html>.
[11] XEP-0407: Mediated Information eXchange (MIX): Miscellaneous Capabilities <https://xmpp.org/extensions/xep-0407.html>.

7. MIX-MUC. Mediated Information eXchange (MIX): Co-existence with MUC (XEP-0408) [12]. This defines how MIX and MUC can be used together.

8. RELIABLE-DELIVERY. MIX-CORE needs messages to be distributed without loss. This specification is important for MIX, but may be useful in other places.

The following table shows which of these specification is mandatory or optional for a MIX server, a server supporting MIX users, a general purpose end user client, and a client providing MIX channel administration.

Specification	MIX Server	Server supporting MIX Clients	End User MIX Client	Administrative MIX Client
MIX-CORE	Mandatory	n/a	Mandatory	Mandatory
MIX-PRESENCE	Optional	n/a	Optional	Optional
MIX-PAM	n/a	Mandatory	Mandatory	Mandatory
MIX-ADMIN	Optional	n/a	n/a	Mandatory
MIX-ANON	Optional	n/a	n/a	Optional
MIX-MISC	Optional	n/a	Optional	Optional
MIX-MUC	Optional	n/a	Optional	n/a
RELIABLE-DELIVERY	Optional	Optional	n/a	n/a

4 Concepts

4.1 Comparison to MUC

This section is written as an introduction to MIX for those with detailed knowledge of Multi-User Chat (XEP-0045) [13], to summarize key differences and some rationale for the differences. For those unfamiliar with MUC, this section should be ignored.
In MUC, a client joins a MUC room. After this, the client is sent history information, presences, and messages until the client leaves the room by going offline. Key MIX features as summarized below:

1. MIX has ”channels”, which are equivalent to MUC rooms.

2. MIX separates out various services, in particular messages and presence. A MIX channel is implemented as a set of PubSub nodes, and a user (not client) can subscribe to a set of nodes. This control means that users can subscribe to presence and/or messages, which

[12] XEP-0408: Mediated Information eXchange (MIX): Co-existence with MUC <https://xmpp.org/extensions/xep-0408.html>.
[13] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

gives useful flexibility. This addresses requirements 3 and 5. Subscribing to message and presence nodes gives a service broadly equivalent to MUC.

3. Messages and presence sent by a MIX channel use the same formats as MUC and do not use PubSub encodings.

4. Channels do not have a ”subject”. This MUC capability is not supported by core MIX.

5. Users join MIX channels for an extended period and participation is not impacted by clients going online and offline (requirement 1). This means that a MIX client uses channel subscriptions as negotiated by the MIX user.

6. MIX messages and presence are always sent and are addressed to the user (bare JID). This addressing is a consequence of users (not clients) being the participants in a MIX channel; It is a key difference between MUC and MIX. This addressing change means that the user's server needs to have MIX-specific behaviour to correctly distribute arriving messages and presence appropriately to MIX clients; there may be zero or more online clients that support MIX. This server behaviour is specified by MIX.

7. MIX messages are archived using MAM on the user's server. This enables flexible access to channel history, independent of the MIX channel server.

8. A user's roster contains the MIX channels to which the user is subscribed. A client can use a MIX roster extension to determine which MIX channels the user is subscribed to. When a client comes online, this will lead to directed presence for the client to be sent to each MIX channel. A MIX channel can then share the user's presence with channel participants that have subscribed to the presence. The MIX channel will also return to the client a full list of presence information for the channel. This means that when a client comes online, it will receive presence updates for the participants in all subscribed MIX channels.

9. In MIX, a Nick belongs to the user and not to each client.

4.2 Specification Approach

MIX will enable a wide range of auxiliary services. The goal of the MIX family of specification is to set out the core capabilities needed for MIX. It is anticipated that additional XEPs will be written to extend the current MIX specification, and the MIX specification family notes some areas where this may happen. Profiles referencing sets of related MIX XEPs may be developed in the future.

4.3 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) [14]. 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) [15] 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) [16] (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:info' 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.

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.

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

12. Each participant has a Stable Participant ID. This is used in some derived JIDs to provide a stable participant reference. It is used to hide JIDs in MIX-ANON.

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.

4.4 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) [17] 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) [18] 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.

4.5 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 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.

4.6 Stable Participant ID

Every channel participant is identified by a Stable Participant ID, which uniquely identifies a channel participant and never changes. The Stable Participant ID MUST NOT contain the '#', '/' or '@' characters.

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

A Participant's Stable Participant ID is defined when a participant joins a channel. While a user is a participant in a Channel, the Stable Participant ID MUST NOT be changed. This mapping between Participant and Stable Participant ID MUST be maintained after the participant leaves the channel. Stable Participant ID values MUST NOT be re-used. If a participant that left a channel joins the channel again, the same Stable Participant ID MAY be used again or a different Stable Participant ID MAY be assigned.

4.7 Standard Nodes

MIX defines a number of standard nodes, and this specification defines three of these nodes and the framework for adding further nodes.

Name	Node	Description	Update	Distribution
Messages	'urn:xmpp:mix:nodes:messages'	For distributing messages to the channel. Each item of this node will contain a message sent to the channel.	Message	Message
Participants	'urn:xmpp:mix:nodes:participants'	For storing 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
Information	'urn:xmpp:mix:nodes:info'	For storing general channel information, such as description.	PubSub	PubSub

Participants is the only mandatory MIX node for a channel, which defines the set of clients that have joined the channel. All other nodes are OPTIONAL. 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.

4.7.1 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-CORE specification. Archiving of other nodes is OPTIONAL.

4.7.2 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.

4.7.3 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 Stable Participant ID of the participant. For example '123456' might name the node item associated with participant 'hag66@shakespeare.example'. Information is stored in a <participant/> element qualified by the 'urn:xmpp:mix:core:1' namespace.
A Nick MAY be associated with a participant, which provides a user-oriented description of the participant. The nick associated with the user 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 (where nicks have been assigned).
A channel MAY require nicks to be mandatory for all participants. This is the default behaviour, and nicks MUST only be optional when this is explicitly configured for a channel as specified in MIX-ADMIN.
Where a nick is provided for a user, it is generally recommended to use this nick to display the user. This enables consistent representation of participants for all participants in the channel.
The real JID of the user MAY be held in the participants node. When the real JID is not present, the procedures defined in MIX-ANON must be followed. Note that only the real JID may be held in participants node. Any JID derived from the stable ID MUST NOT be held. The user's JID is stored in a <jid/> child element of the <participant/> element.
When a user joins a channel, an item representing the user is added to the participants node by the MIX service. When a user leaves a channel, the user's item is removed from the participants node. The participants node MUST NOT be directly modified using pubsub.
It may be useful for clients to read the participants list. However it is not necessary for message and presence display, as both messages and presence contain sufficient information

to enable display.
Users MAY subscribe to, and read information from 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 MANDATORY. 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'>
    <participant xmlns='urn:xmpp:mix:core:1'>
      <nick>thirdwitch</nick>
      <jid>hag66@shakespeare.example</jid>
    </participant>
  </item>
</items>
```

4.7.4 Information Node

The Information node holds information about the channel. It will often be helpful for MIX clients to be able to display this information. 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	-	-
'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	-	-

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. The format of the Information node follows Data Forms (XEP-0004) [19]. 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 2: 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'>
        <value>urn:xmpp:mix:core: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>
```

4.8 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.

5 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

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

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 [20] 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 [21] containing an error defined in RFC 6120 [22], which is the same error set as for IQs.
3. PubSub. Where MIX protocol messages use PubSub protocol, the errors defined in Publish-Subscribe (XEP-0060) [23] MUST be used and supported.

6 Discovery

6.1 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 3: 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'/>
```

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

```
  </query>
</iq>
```

To determine if a domain hosts a MIX service, a Service Discovery (XEP-0030) [24] info query is sent in the usual manner to determine capabilities.

Listing 4: 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:core:1' feature, and the identity MUST have a category of 'conference' and a type of 'mix', as shown in the following example:

Listing 5: Service responds with Disco Info result

```
<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='mix'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='urn:xmpp:mix:core:1'/>
    <feature var='urn:xmpp:mix:core:1#searchable'>
  </query>
</iq>
```

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

- 'urn:xmpp:mix:core:1': This indicates support of MIX, and is returned by all MIX services.
- 'urn:xmpp:mix:core:1#searchable': This is shown in the above example and indicates that a the MIX Service MAY be searched for channels. The presence of this feature can be used by a client to guide the user to search for channels in a MIX service.

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

- 'urn:xmpp:mix:core: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 a 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) [25], 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) [26], as although MIX makes use of PubSub it is not a generic PubSub service.

6.2 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 6: Client Queries for Channels on MIX Service

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

Listing 7: 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>
```

6.3 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.

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

Listing 8: 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'/>
</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.
Note that a node MAY be both a MIX channel and a MUC room. In this case, the node will return both MIX and MUC information. MIX and MUC clients MUST be able to handle this.

Listing 9: Channel Returns Disco Info Result

```
<iq from='coven@mix.shakespeare.example'
    id='ik3vs715'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='A␣Dark␣Cave'
        type='mix'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='urn:xmpp:mix:core:1'/>
    <feature var='urn:xmpp:mam:2'/>
  </query>
</iq>
```

6.4 Discovering Nodes at a Channel

Use disco#items to find the nodes associated with a channel. The MIX service MUST only return nodes that exist and that the user making the query has rights to subscribe to.
Discovering nodes in MIX MUST use a node='mix' attribute in this query. This is to facilitate server implementations that support a single node being both a MIX channel and a MUC room. MUC rooms use this query to return a list of occupants. The node='mix' attribute allows a server to support both MIX and MUC queries without requiring any change to MUC clients. Where a node only supports a MIX channel, the server MUST reject queries without a node='mix' attribute.

Listing 10: 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 11: 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>
```

6.5 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 12: Client Requests Channel Information

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

Listing 13: MIX Service Returns Channel Information

```
<iq from='coven@mix.shakespeare.example'
    id='kl2fax27'
    to='hag66@shakespeare.example/UUID-c8y/1573'
    type='result'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
```

```
    <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'>
            <value>urn:xmpp:mix:core: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>
```

6.6 Determining the Participants in a Channel

Participants in the channel are determined using PubSub retrieval of the Participants Node which will give Stable Participant ID, JID and nick. Clients using a channel MAY determine participants on start-up, to enable display of participants.

Listing 14: User's Client Requests Participant List

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

Listing 15: MIX Service Returns Participant List

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

```
        <participant xmlns='urn:xmpp:mix:core:1'>
          <nick>thirdwitch</nick>
          <jid>hag66@shakespeare.example</jid>
        </participant>
      </item>
      <item id='87123'>
        <participant xmlns='urn:xmpp:mix:core:1'>
          <nick>top witch</nick>
          <jid>hecate@shakespeare.example</jid>
        </participant>
      </item>
    </items>
  </pubsub>
</iq>
```

6.7 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 16: 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'>
    <identity
        category='client'
        name='Swift'
        type='pc'/>
    <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:core:1'/>
  </query>
```

```
</iq>
```

7 Use Cases

7.1 Common User Use Cases

7.1.1 Model for Join and Leave

MIX Join and Leave communication goes through the client's server. The full specification of client interaction with the client's server is specified in MIX-PAM. This specification shows the protocol between the user's server and the MIX server and sets out behaviour of the MIX server.

7.1.2 Joining a Channel

A user joins a channel by sending a MIX "join" command from one of the user's clients, which will be relayed to the server as specified in MIX-PAM. 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. Note that the presence node is specified in MIX-PRESENCE. The <join/> is a child element of <iq/> element. The <join/> element is qualified by the 'urn:xmpp:mix:core:1' namespace. The requested nodes are encoded as <subscribe/> child elements of the <join/> element. A nick MAY be specified as a <nick/> 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 to some nodes. 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, as specified in MIX-PAM. This means that the join request to the MIX service will come from the user's server from the user's bare JID.

Listing 17: 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:core: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:info'/>
    <nick>third witch</nick>
  </join>
</iq>
```

The channel responds to the user's sever with an IQ-result addressed to the user's bare JID, which will be processed as specified in MIX-PAM. 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 user's Stable Participant ID is returned as an 'id' attribute in the join. The following example shows the result of the above request when the request is completed in full.

Listing 18: 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:core:1'
        id='123456'>
    <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:info'/>
    <nick>third witch</nick>
  </join>
</iq>
```

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.

Listing 19: 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:core:1'
        id='123456'>
```

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

After a successful join and before sending the response, the channel MUST subscribe the user to the negotiated nodes and adds the user to the participants node. When these changes are made, the MIX channel MUST send a PubSub notification of the change to subscribers of the participants node. The following example shows such a notification.

Listing 20: Channel Distributes New Participant Information

```
<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'>
        <participant xmlns='urn:xmpp:mix:core:1'>
          <jid>hag66@shakespeare.example</jid>
          <nick>third witch</nick>
        </participant>
      </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 Stable Participant ID of the new channel participant. The <participant> element MUST include a <jid> element with the JID of the participant, unless MIX-ANON is being followed to hide participant JIDs. The <nick> element will not be included at this point, unless it is automatically generated by the channel. In the likely event that a Nick is subsequently added, an update with the <nick> element will be distributed.
A user MAY subsequently modify subscription to nodes in a channel by sending a subscription modification request encoded as a <update-subscription/> child element of <iq/> element. The <update-subscription/> element is qualified by the 'urn:xmpp:mix:core:1' namespace. The requested nodes are encoded as <subscribe/> and <unsubscribe/> child elements of the <update-subscription/> element with the node name encoded as a 'node' attribute. This modification goes directly from client to the 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 21: 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:core:1'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <unsubscribe node='urn:xmpp:mix:nodes:presence'/>
  </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:core:1'
                       jid='hag66@shakespeare.example'>
    <subscribe node='urn:xmpp:mix:nodes:messages'/>
    <unsubscribe node='urn:xmpp:mix:nodes:presence'/>
  </update-subscription>
</iq>
```

A user MAY specify a nick when joining the channel. Channels MAY have mandatory nicks, which is default behavior. Therefore it will generally be necessary to include a nick when joining a channel. If nick is missing on a channel where nick is mandatory, the join MUST be rejected. Other error cases are dealt with below with the nick management commands. Where a nick is specified, the join will only complete if the nick is accepted. The nick used MUST be reported back in the join result.

7.1.3 Leaving a Channel

Users generally remain in a channel for an extended period of time. In particular the user remains as a participant of the channel when the user goes offline. Note that this is different to Multi-User Chat (XEP-0045) [27], 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, the user's server sends a MIX "leave" command to the channel, as specified in MIX-PAM. The leave command is encoded as a <leave/> child element of <iq/> element. The <leave/> element is qualified by the 'urn:xmpp:mix:core:1' namespace. The following example shows a leave request to a MIX channel.

Listing 22: 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:core:1'/>
</iq>
```

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

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

Listing 23: 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:core: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 list. Presence removal is specified in MIX-PRESENCE. Deletion from the participants node functions as if the item (channel participant) had been deleted using the PubSub retract mechanism with notification set to true. Notification of the participant deletion is sent to clients subscribed to the participants PubSub node using PubSub protocol, with the node identified by the stable id, as shown in the example below.

Listing 24: Participant Node Subscriber is Notified of Participant Removal

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

7.1.4 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. 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.

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:core: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 25: 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:core: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 26: 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:core:1'>
    <nick>thirdwitch</nick>
  </setnick>
</iq>
```

7.1.5 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 on the client's server. There are three basic approaches a client will take:

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

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) [29], 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 MIX channel.

7.1.6 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) [30]. Messages are sent directly to the MIX channel from the user's client. The message id is selected by the client.

Listing 27: 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>
```

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

1. A <nick> element that contains the Nick of the message sender, taken from the Participants Node. This MUST be present if a Nick is defined for the user.
2. A <jid> element containing the real JID of the sender. This MUST be present, unless following the "JID Hidden" model defined in MIX-ANON. If this element is omitted, the <nick> element MUST be present.

[29] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
[30] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

MIX messages are distributed by the channel with the from using the JID of the channel, with the Stable Participant ID of the sender in the resource. This enables a receiving system to distinguish messages based on sender using only the JID.
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 as specified in MIX-PAM. 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 28: 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:core:1'>
    <nick>thirdwitch</nick>
    <jid>hag66@shakespeare.example</jid>
  </mix>
</message>
```

Listing 29: Channel Reflects Message to Participants

```
<message from='coven@mix.shakespeare.example/123456'
         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:core:1'>
    <nick>thirdwitch</nick>
    <jid>hag66@shakespeare.example</jid>
  </mix>
</message>
```

The message originator may wish to correlate the reflected message with the submitted message. To do this, the originator should include an <origin-id> element in the message as specified in Unique and Stable Stanza IDs (XEP-0359) [31].

7.2 Use of MAM

MIX channel nodes MAY be archived. In order to provide a service equivalent to MUC, it is necessary to archive messages sent to the channel. It is anticipated the most MIX services will

[31] XEP-0359: Unique and Stable Stanza IDs <https://xmpp.org/extensions/xep-0359.html>.

archive at least messages using MAM.

7.2.1 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.

7.2.2 Retrieving Messages

The client's local server MAY archive messages and advertise this capability as specified in Mediated Information eXchange (MIX): Participant Server Requirements (XEP-0405) [32]. If this is done, clients MUST 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. This enables new channel participants to access the historical archives.

7.2.3 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.

7.3 Administrative Use Cases

7.3.1 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:core:1#create-channel' feature is returned, the user is able to create a channel.

[32] XEP-0405: Mediated Information eXchange (MIX): Participant Server Requirements <https://xmpp.org/extensions/xep-0405.html>.

Listing 30: 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='mix'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='urn:xmpp:mix:core:1'/>
    <feature var='urn:xmpp:mix:core:1#create-channel'>
  </query>
</iq>
```

7.3.2 Creating a Channel

A client creates a channel by sending a simple request to the MIX service. A channel is always 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:core: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 31: 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:core: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:core:1'/>
```

```
</iq>
```

When a channel is created , the Owner in the configuration is set to the JID that creates the channel. Modifying channel parameters is specified in MIX-ADMIN. Consideration should be given to selection of default parameters. It will typically be desirable to create channels with restrictive default settings that the owner MAY choose to relax.

7.3.3 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 32: 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:core: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:core:1'/>
</iq>
```

7.3.4 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) [33] 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; This is specified in MIX-ADMIN. 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:core:1' namespace. The <destroy/> element MUST have a 'channel' attribute to specify the channel to be destroyed. A client

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

destroys a channel using a simple set operation, as shown in the following example.

Listing 33: Client Destroys a Channel

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

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

7.3.5 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.

8 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) [34]. These capabilities will not be added to core MIX but they could in the future be specified as independent XEPs.

8.1 Password Controlled Channels

Multi-User Chat (XEP-0045) [35] 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.

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

8.2 Voice Control

Multi-User Chat (XEP-0045) [36] 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.

8.3 Subject

Multi-User Chat (XEP-0045) [37] 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.

9 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 [38].

10 Security Considerations

MIX is built over MAM and PubSub and the security considerations of Message Archive Management (XEP-0313) [39] and Publish-Subscribe (XEP-0060) [40] MUST be considered. These services protect MIX channel information, which can be sensitive and needs appropriate protection.
There is no MIX equivalent to Multi-User Chat (XEP-0045) [41] password controlled rooms, which avoids a number of security issues.

[36] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[37] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[38] RFC 7700: Preparation, Enforcement, and Comparison of Internationalized Strings Representing Nicknames<http://tools.ietf.org/html/rfc7700>.
[39] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
[40] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[41] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

MIX-ADMIN defines flexible access control options, which MUST be used in a manner appropriate to the security requirements of MIX users and services.

11 IANA Considerations

None.

12 XMPP Registrar Considerations

The urn:xmpp:mix namespace needs to be registered.
The conference type 'mix' needs to be registered.

13 XML Schema

To be supplied when MIX progresses to proposed standard.

14 Acknowledgements

Peter St Andre provided key early input to MIX and worked on the original specification with Kevin Smith.
Thanks to the following who have made contributions to the ongoing MIX specification development: W. Martin Borgert, Dave Cridland, Daniel Gultsch, Tarun Gupta, Philipp Hancke, Waqas Hussain, Timothée Jaussoin, Evgeny Khramtsov, Georg Lukas, Tobias Markmann, Ralph Meijer, Edwin Mons, Emmanuel Gil Peyrot, Manuel Rubio, Florian Schmaus, Lance Stout, Sam Whited, Jonas Wielicki, Matthew Wild and one anonymous reviewer.