XEP-0045: Multi-User Chat

Peter Saint-Andre
mailto:stpeter@stpeter.im
xmpp:stpeter@jabber.org
https://stpeter.im/

2024-09-17
Version 1.35.1

Status	Type	Short Name
Draft	Standards Track	muc

This specification defines an XMPP protocol extension for multi-user text chat, whereby multiple XMPP users can exchange messages in the context of a room or channel, similar to Internet Relay Chat (IRC). In addition to standard chatroom features such as room topics and invitations, the protocol defines a strong room control model, including the ability to kick and ban users, to name room moderators and administrators, to require membership or passwords in order to join the room, etc.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Traditionally, instant messaging is thought to consist of one-to-one chat rather than many-to-many chat, which is called variously ”groupchat” or ”text conferencing”. Groupchat functionality is familiar from systems such as Internet Relay Chat (IRC) and the chatroom functionality offered by popular consumer IM services. The Jabber/XMPP community developed and implemented a basic groupchat protocol as long ago as 1999. That ”groupchat 1.0” (GC) protocol provided a minimal feature set for chat rooms but was rather limited in scope. This specification (Multi-User Chat or MUC) is not compatible to the groupchat 1.0 protocol, but provides advanced features such as invitations, room moderation and administration, and specialized room types.

2 Scope

This document addresses common requirements related to configuration of, participation in, and administration of individual text-based conference rooms. All of the requirements addressed herein apply at the level of the individual room and are ”common” in the sense that they have been widely discussed within the Jabber/XMPP community or are familiar from existing text-based conference environments (e.g., Internet Relay Chat as defined in RFC 1459 [1] and its successors: RFC 2810 [2], RFC 2811 [3], RFC 2812 [4], RFC 2813 [5]).
This document explicitly does *not* address the following:

- Relationships between rooms (e.g., hierarchies of rooms)
- Management of multi-user chat services (e.g., managing permissions across an entire service or registering a global room nickname); such use cases are specified in Service Administration (XEP-0133) [6]
- Moderation of individual messages
- Encryption of messages sent through a room
- Advanced features such as attaching files to a room, integrating whiteboards, and using MUC rooms as a way to manage the signalling for multi-user audio or video conferencing (see Multiparty Jingle (XEP-0272) [7])
- Interaction between MUC deployments and foreign chat systems (e.g., gateways to IRC or to legacy IM systems)

[1] RFC 1459: Internet Relay Chat <http://tools.ietf.org/html/rfc1459>.
[2] RFC 2810: Internet Relay Chat: Architecture <http://tools.ietf.org/html/rfc2810>.
[3] RFC 2811: Internet Relay Chat: Channel Management <http://tools.ietf.org/html/rfc2811>.
[4] RFC 2812: Internet Relay Chat: Client Protocol <http://tools.ietf.org/html/rfc2812>.
[5] RFC 2813: Internet Relay Chat: Server Protocol <http://tools.ietf.org/html/rfc2813>.
[6] XEP-0133: Service Administration <https://xmpp.org/extensions/xep-0133.html>.
[7] XEP-0272: Multiparty Jingle <https://xmpp.org/extensions/xep-0272.html>.

- Mirroring or replication of rooms among multiple MUC deployments

This limited scope is not meant to disparage such topics, which are of inherent interest; however, it is meant to focus the discussion in this document and to present a comprehensible protocol that can be implemented by client and service developers alike. Future specifications might address the topics mentioned above.

3 Requirements

This document addresses the minimal functionality provided by Jabber-based multi-user chat services that existed in 2002 when development of MUC began. This design is based on the original groupchat 1.0 protocol, with the result that:

- Each room is identified as a "room JID" <room@service> (e.g., <jdev@conference.jabber.org>), where "room" is the name of the room and "service" is the hostname at which the multi-user chat service is running.
- Each occupant in a room is identified as an "occupant JID" <room@service/nick>, where "nick" is the room nickname of the occupant as specified on entering the room or subsequently changed during the occupant's visit.
- A user enters a room (i.e., becomes an occupant) by sending directed presence to <room@service/nick>.
- An occupant can change his or her room nickname and availability status within the room by sending presence information to <room@service/newnick>.
- Messages sent within multi-user chat rooms are of a special type "groupchat" and are addressed to the room itself (room@service), then reflected to all occupants.
- An occupant exits a room by sending presence of type "unavailable" to its current <room@service/nick>.

The additional features and functionality addressed in MUC include the following:

1. native conversation logging (no in-room bot required)
2. enabling users to request membership in a room
3. enabling occupants to view an occupant's full JID in a non-anonymous room
4. enabling moderators to view an occupant's full JID in a semi-anonymous room
5. allowing only moderators to change the room subject
6. enabling moderators to kick participants and visitors from the room

7. enabling moderators to grant and revoke voice (i.e., the privilege to speak) in a moderated room, and to manage the voice list
8. enabling admins to grant and revoke moderator status, and to manage the moderator list
9. enabling admins to ban users from the room, and to manage the ban list
10. enabling admins to grant and revoke membership privileges, and to manage the member list for a members-only room
11. enabling owners to configure various room parameters (e.g., limiting the number of occupants)
12. enabling owners to specify other owners
13. enabling owners to grant and revoke admin status, and to manage the admin list
14. enabling owners to destroy the room

In addition, this document provides protocol elements for supporting the following room types:

1. public vs. hidden
2. persistent vs. temporary
3. password-protected vs. unsecured
4. members-only vs. open
5. moderated vs. unmoderated
6. non-anonymous vs. semi-anonymous

The extensions needed to implement these requirements are qualified by the 'http://jabber.org/protocol/muc' namespace (and the #owner, #admin, and #user fragments on the main namespace URI).

4 Terminology

4.1 General Terms

Affiliation A long-lived association or connection with a room; the possible affiliations are "owner", "admin", "member", and "outcast" (naturally it is also possible to have no affiliation); affiliation is distinct from role. An affiliation lasts across a user's visits to a room.

Ban To remove a user from a room such that the user is not allowed to re-enter the room (until and unless the ban has been removed). A banned user has an affiliation of "outcast".

Bare JID The <user@host> by which a user is identified outside the context of any existing session or resource; contrast with Full JID and Occupant JID.

Full JID The <user@host/resource> by which an online user is identified outside the context of a room; contrast with Bare JID and Occupant JID.

GC The minimal "groupchat 1.0" protocol developed within the Jabber community in 1999; Old versions of MUC were backwards-compatible with GC.

History A limited number of message stanzas sent to a new occupant to provide the context of current discussion.

Invitation A special message sent from one user to another asking the recipient to join a room; the invitation can be sent directly (see Direct MUC Invitations (XEP-0249) XEP-0249: Direct MUC Invitations <https://xmpp.org/extensions/xep-0249.html>.) or mediated through the room (as described under Inviting Another User to a Room).

IRC Internet Relay Chat.

Kick To temporarily remove a participant or visitor from a room; the user is allowed to re-enter the room at any time. A kicked user has a role of "none".

Logging Storage of discussions that occur within a room for public retrieval outside the context of the room.

Member A user who is on the "permitted" list for a members-only room or who is registered with an open room. A member has an affiliation of "member".

Moderator A room role that is usually associated with room admins but that can be granted to non-admins; is allowed to kick users, grant and revoke voice, etc. A moderator has a role of "moderator".

MUC The multi-user chat protocol for text-based conferencing specified in this document.

Multi-Session Nick If allowed by the service, a user can associate more than one full JID with the same occupant JID (e.g., the user juliet@capulet.lit is allowed to log in simultaneously as the nick "JuliC" in the characters@chat.shakespeare.lit chatroom from both juliet@capulet.lit/balcony and juliet@capulet.lit/chamber). Multi-session nicks are not currently defined in this document.

Occupant Any user who is in a room (this is an "abstract class" and does not correspond to any specific role).

Occupant JID The <room@service/nick> by which an occupant is identified within the context of a room; contrast with Bare JID and Full JID.

Outcast A user who has been banned from a room. An outcast has an affiliation of ”outcast”.

Participant An occupant who does not have admin status; in a moderated room, a participant is further defined as having voice (in contrast to a visitor). A participant has a role of ”participant”.

Private Message A message sent from one occupant directly to another’s occupant JID (not to the room itself for broadcasting to all occupants).

Role A temporary position or privilege level within a room, distinct from a user’s long-lived affiliation with the room; the possible roles are ”moderator”, ”participant”, and ”visitor” (it is also possible to have no defined role). A role lasts only for the duration of an occupant’s visit to a room.

Room A virtual space that users figuratively enter in order to participate in real-time, text-based conferencing with other users.

Room Administrator A user empowered by the room owner to perform administrative functions such as banning users; however, a room administrator is not allowed to change the room configuration or to destroy the room. An admin has an affiliation of ”admin”.

Room ID The localpart of a Room JID, which might be opaque and thus lack meaning for human users (see under Business Rules for syntax); contrast with Room Name.

Room JID The <room@service> address of a room.

Room Name A user-friendly, natural-language name for a room, configured by the room owner and presented in Service Discovery queries; contrast with Room ID.

Room Nickname The resourcepart of an Occupant JID (see Business Rules for syntax); this is the ”friendly name” by which an occupant is known in the room.

Room Owner The user who created the room or a user who has been designated by the room creator or owner as someone with owner status (if allowed); an owner is allowed to change the room configuration and destroy the room, in addition to all admin status. An owner has an affiliation of ”owner”.

Room Roster A client’s representation of the occupants in a room.

Server An XMPP server that may or may not have associated with it a text-based conferencing service.

Service A host that offers text-based conferencing capabilities; often but not necessarily a sub-domain of an XMPP server (e.g., conference.jabber.org).

Subject A temporary discussion topic within a room.

Visit A user’s ”session” in a room, beginning when the user enters the room (i.e., becomes an occupant) and ending when the user exits the room.

Visitor In a moderated room, an occupant who does not have voice (in contrast to a participant). A visitor has a role of "visitor".

Voice In a moderated room, the privilege to send messages to all occupants.

4.2 Room Types

Hidden Room A room that cannot be found by any user through normal means such as searching and service discovery; antonym: Public Room.

Members-Only Room A room that a user cannot enter without being on the member list; antonym: Open Room.

Moderated Room A room in which only those with "voice" are allowed to send messages to all occupants; antonym: Unmoderated Room.

Non-Anonymous Room A room in which an occupant's full JID is exposed to all other occupants, although the occupant can request any desired room nickname; contrast with Semi-Anonymous Room.

Open Room A room that non-banned entities are allowed to enter without being on the member list; antonym: Members-Only Room.

Password-Protected Room A room that a user cannot enter without first providing the correct password; antonym: Unsecured Room.

Persistent Room A room that is not destroyed if the last occupant exits; antonym: Temporary Room.

Public Room A room that can be found by any user through normal means such as searching and service discovery; antonym: Hidden Room.

Semi-Anonymous Room A room in which an occupant's full JID can be discovered by room admins only; contrast with Non-Anonymous Room.

Temporary Room A room that is destroyed if the last occupant exits; antonym: Persistent Room.

Unmoderated Room A room in which any occupant is allowed to send messages to all occupants; antonym: Moderated Room.

Unsecured Room A room that anyone is allowed to enter without first providing the correct password; antonym: Password-Protected Room.

4.3 Dramatis Personae

Most of the examples in this document use the scenario of the witches' meeting held in a dark cave at the beginning of Act IV, Scene I of Shakespeare's Macbeth, represented here as the "coven@chat.shakespeare.lit" chatroom. The characters are as follows:

Room Nickname	Full JID	Affiliation
firstwitch	crone1@shakespeare.lit/desktop	Owner
secondwitch	wiccarocks@shakespeare.lit/laptop	Admin
thirdwitch	hag66@shakespeare.lit/pda	None

5 Roles, Affiliations, and Privileges

A user might be allowed to perform any number of actions in a room, from joining or sending a message to changing configuration options or destroying the room altogether. We call each permitted action a ”privilege”. There are two ways we might structure privileges:

1. Define each privilege atomically and explicitly define each user’s particular privileges; this is flexible but can be confusing to manage.

2. Define bundles of privileges that are generally applicable and assign a user-friendly ”shortcut” to each bundle (e.g., ”moderator” or ”admin”).

MUC takes the second approach.
MUC also defines two different associations: long-lived affiliations and session-specific roles. These two association types are distinct from each other in MUC, since an affiliation lasts across visits, while a role lasts only for the duration of a visit. In addition, there is no one-to-one correspondence between roles and affiliations; for example, someone who is not affiliated with a room may be a (temporary) moderator, and a member may be a participant or a visitor in a moderated room. These concepts are explained more fully below.

5.1 Roles

The following roles are defined:

Name	Support
Moderator	REQUIRED
None	N/A (the absence of a role)
Participant	REQUIRED
Visitor	RECOMMENDED

Roles are temporary in that they do not necessarily persist across a user's visits to the room and MAY change during the course of an occupant's visit to the room. An implementation MAY persist roles across visits and SHOULD do so for moderated rooms (since the distinction between visitor and participant is critical to the functioning of a moderated room).
There is no one-to-one mapping between roles and affiliations (e.g., a member could be a participant or a visitor).
A moderator is the most powerful role within the context of the room, and can to some extent manage other occupants' roles in the room. A participant has fewer privileges than a moderator, although he or she always has the right to speak. A visitor is a more restricted role within the context of a moderated room, since visitors are not allowed to send messages to all occupants (depending on room configuration, it is even possible that visitors' presence will not be broadcasted to the room).
Roles are granted, revoked, and maintained based on the occupant's room nickname or full JID rather than bare JID. The privileges associated with these roles, as well as the actions that trigger changes in roles, are defined below.
Information about roles MUST be sent in all presence stanzas generated or reflected by the room and thus sent to occupants (if the room is configured to broadcast presence for a given role).

5.1.1 Privileges

For the most part, roles exist in a hierarchy. For instance, a participant can do anything a visitor can do, and a moderator can do anything a participant can do. Each role has all the privileges possessed by the next-lowest role, plus additional privileges; these privileges are specified in the following table as defaults (an implementation MAY provide configuration options that override these defaults).

Privilege	None	Visitor	Participant	Moderator
Present in Room	No	Yes	Yes	Yes
Receive Messages	No	Yes	Yes	Yes
Receive Occupant Presence	No	Yes	Yes	Yes
Broadcast Presence to All Occupants	No	Yes*	Yes	Yes
Change Availability Status	No	Yes*	Yes	Yes
Change Room Nickname	No	Yes*	Yes	Yes
Send Private Messages	No	Yes*	Yes	Yes
Invite Other Users	No	Yes*	Yes*	Yes
Send Messages to All	No	No**	Yes	Yes
Modify Subject	No	No*	Yes*	Yes
Kick Participants and Visitors	No	No	No	Yes
Grant Voice	No	No	No	Yes
Revoke Voice	No	No	No	Yes***

* Default; configuration settings MAY modify this privilege.
* An implementation MAY grant voice by default to visitors in unmoderated rooms.
** A moderator MUST NOT be able to revoke voice privileges from an admin or owner.

5.1.2 Default Roles

The following table summarizes the initial default roles that a service SHOULD set based on the user's affiliation (there is no role associated with the "outcast" affiliation, since such users are not allowed to enter the room).

Room Type	None	Member	Admin	Owner
Moderated	Visitor	Participant	Moderator	Moderator
Unmoderated	Participant	Participant	Moderator	Moderator
Members-Only	N/A *	Participant	Moderator	Moderator
Open	Participant	Participant	Moderator	Moderator

* Entry is not permitted.

5.1.3 Changing Roles

The ways in which an occupant's role changes are well-defined. Sometimes the change results from the occupant's own action (e.g., entering or exiting the room), whereas sometimes the change results from an action taken by a moderator, admin, or owner. If an occupant's role changes, a MUC service implementation MUST change the occupant's role to reflect the change and communicate the change to all occupants (if the room is configured to broadcast presence from entities with a given role). Role changes and their triggering actions are specified in the following table.

>	None	Visitor	Participant	Moderator
None	--	Enter moderated room	Enter unmoderated room	Admin or owner enters room
Visitor	Exit room or be kicked by a moderator	--	Moderator grants voice	Admin or owner grants moderator status
Participant	Exit room or be kicked by a moderator	Moderator revokes voice	--	Admin or owner grants moderator status

>	None	Visitor	Participant	Moderator
Moderator	Exit room or be kicked by an admin or owner *	Admin or owner changes role to visitor *	Admin or owner changes role to participant or revokes moderator status *	--

* A moderator SHOULD NOT be allowed to revoke moderation privileges from someone with a higher affiliation than themselves (i.e., an unaffiliated moderator SHOULD NOT be allowed to revoke moderation privileges from an admin or an owner, and an admin SHOULD NOT be allowed to revoke moderation privileges from an owner).
Note: Certain roles are typically implicit in certain affiliations. For example, an admin or owner is automatically a moderator, so if an occupant is granted an affiliation of admin then the occupant will by that fact be granted a role of moderator; similarly, when an occupant is granted an affiliation of member in a moderated room, the occupant automatically has a role of participant. However, the loss of the admin affiliation does not necessarily mean that the occupant no longer has a role of moderator (since a ”mere” occupant can be a moderator). Therefore, the role that is gained when an occupant is granted a certain affiliation is stable, whereas the role that is lost when an occupant loses a certain affilitation is not hardcoded and is left up to the implementation.

5.2 Affiliations

The following affiliations are defined:

1. Owner
2. Admin
3. Member
4. Outcast
5. None (the absence of an affiliation)

Support for the owner affiliation is REQUIRED. Support for the admin, member, and outcast affiliations is RECOMMENDED. (The ”None” affiliation is the absence of an affiliation.)
These affiliations are long-lived in that they persist across a user’s visits to the room and are not affected by happenings in the room. In addition, there is no one-to-one mapping between these affiliations and an occupant’s role within the room. Affiliations are granted, revoked, and maintained based on the user’s bare JID, not the nick as with roles.
If a user without a defined affiliation enters a room, the user’s affiliation is defined as ”none”; however, this affiliation does not persist across visits (i.e., a service does not maintain a ”none

list" across visits).
The member affiliation provides a way for a room owner or admin to specify a "permitted" list of users who are allowed to enter a members-only room. When a member enters a members-only room, his or her affiliation does not change, no matter what his or her role is. The member affiliation also provides a way for users to register with an open room and thus be lastingly associated with that room in some way (one result might be that the service could reserve the user's nickname in the room).
An outcast is a user who has been banned from a room and who is not allowed to enter the room.
Information about affiliations MUST be sent in all presence stanzas generated or reflected by the room and sent to occupants (if the room is configured to broadcast presence from entities with a given role).

5.2.1 Privileges

For the most part, affiliations exist in a hierarchy. For instance, an owner can do anything an admin can do, and an admin can do anything a member can do. Each affiliation has all the privileges possessed by the next-lowest affiliation, plus additional privileges; these privileges are specified in the following table.

Privilege	Outcast	None	Member	Admin	Owner
Enter Open Room	No	Yes*	Yes	Yes	Yes
Register with Open Room	No	Yes	N/A	N/A	N/A
Retrieve Member List***	No	No	Yes	Yes	Yes
Enter Members-Only Room	No	No	Yes*	Yes	Yes
Ban Members and Unaffiliated Users	No	No	No	Yes	Yes
Edit Member List	No	No	No	Yes	Yes
Assign and Remove Moderator Role	No	No	No	Yes**	Yes**
Edit Admin List	No	No	No	No	Yes
Edit Owner List	No	No	No	No	Yes
Change Room Configuration	No	No	No	No	Yes
Destroy Room	No	No	No	No	Yes

* As a default, an unaffiliated user enters a moderated room as a visitor, and enters an open room as a participant. A member enters a room as a participant. An admin or owner enters a room as a moderator.
* As noted, a moderator SHOULD NOT be allowed to revoke moderation privileges from someone with a higher affiliation than themselves (i.e., an unaffiliated moderator SHOULD NOT be allowed to revoke moderation privileges from an admin or an owner, and an admin

SHOULD NOT be allowed to revoke moderation privileges from an owner).
** When a room is configured to be semi-anonymous, there clearly is an intent to hide JIDs. In such rooms, members SHOULD NOT be allowed to retrieve the member list (as that list MUST contain the JID of each member).

5.2.2 Changing Affiliations

The ways in which a user's affiliation changes are well-defined. Sometimes the change results from the user's own action (e.g., registering as a member of the room), whereas sometimes the change results from an action taken by an admin or owner. If a user's affiliation changes, a MUC service implementation MUST change the user's affiliation to reflect the change and communicate that to all occupants (if the room is configured to broadcast presence from entities with a given role). Affiliation changes and their triggering actions are specified in the following table.

>	Outcast	None	Member	Admin	Owner
Outcast	--	Admin or owner removes ban	Admin or owner adds user to member list	Owner adds user to admin list	Owner adds user to owner list
None	Admin or owner applies ban	--	Admin or owner adds user to member list, or user registers as member (if allowed)	Owner adds user to admin list	Owner adds user to owner list
Member	Admin or owner applies ban	Admin or owner changes affiliation to "none"	--	Owner adds user to admin list	Owner adds user to owner list
Admin	Owner applies ban	Owner changes affiliation to "none"	Owner changes affiliation to "member"	--	Owner adds user to owner list
Owner	Owner applies ban	Owner changes affiliation to "none"	Owner changes affiliation to "member"	Owner changes affiliation to "admin"	--

6 Entity Use Cases

A MUC implementation MUST support Service Discovery (XEP-0030) [8] ("disco"). Any entity can complete the following disco-related use cases.

6.1 Discovering a MUC Service

An entity often discovers a MUC service by sending a Service Discovery items ("disco#items") request to its own server.

Listing 1: Entity Queries Server for Associated Services

```
<iq from='hag66@shakespeare.lit/pda'
    id='h7ns81g'
    to='shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

The server then returns the services that are associated with it.

Listing 2: Server Returns Disco Items Result

```
<iq from='shakespeare.lit'
    id='h7ns81g'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='chat.shakespeare.lit'
          name='Chatroom Service'/>
  </query>
</iq>
```

6.2 Discovering the Features Supported by a MUC Service

An entity may wish to discover if a service implements the Multi-User Chat protocol; in order to do so, it sends a service discovery information ("disco#info") query to the MUC service's JID.

Listing 3: Entity Queries Chat Service for MUC Support via Disco

```
<iq from='hag66@shakespeare.lit/pda'
    id='lx09df27'
    to='chat.shakespeare.lit'
    type='get'>
```

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

```
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

The service MUST return its identity and the features it supports.

Listing 4: Service Returns Disco Info Result

```
<iq from='chat.shakespeare.lit'
    id='lx09df27'
    to='hag66@shakespeare.lit/pda'
    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'/>
  </query>
</iq>
```

6.3 Discovering Rooms

The service discovery items (”disco#items”) protocol enables an entity to query a service for a list of associated items, which in the case of a chat service would consist of the specific chat rooms hosted by the service.

Listing 5: Entity Queries Chat Service for Rooms

```
<iq from='hag66@shakespeare.lit/pda'
    id='zb8q41f4'
    to='chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

The service SHOULD return a full list of the public rooms it hosts (i.e., not return any rooms that are hidden).

Listing 6: Service Returns Disco Items Result

```
<iq from='chat.shakespeare.lit'
    id='zb8q41f4'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='heath@chat.shakespeare.lit'
          name='A Lonely Heath'/>
```

```
    <item jid='coven@chat.shakespeare.lit'
          name='A Dark Cave'/>
    <item jid='forres@chat.shakespeare.lit'
          name='The Palace'/>
    <item jid='inverness@chat.shakespeare.lit'
          name='Macbeth&apos;s Castle'/>
  </query>
</iq>
```

If the full list of rooms is large (see Service Discovery (XEP-0030) [9] for details), the service MAY return only a partial list of rooms. If it does so, it SHOULD include a <set/> element qualified by the 'http://jabber.org/protocol/rsm' namespace (as defined in Result Set Management (XEP-0059) [10]) to indicate that the list is not the full result set.

Listing 7: Service Returns Limited List of Disco Items Result

```
<iq from='chat.shakespeare.lit'
    id='hx51v49s'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='alls-well-that-ends-well@chat.shakespeare.lit'/>
    <item jid='as-you-like-it@chat.shakespeare.lit'/>
    <item jid='cleopatra@chat.shakespeare.lit'/>
    <item jid='comedy-of-errors@chat.shakespeare.lit'/>
    <item jid='coriolanus@chat.shakespeare.lit'/>
    <item jid='cymbeline@chat.shakespeare.lit'/>
    <item jid='hamlet@chat.shakespeare.lit'/>
    <item jid='henry-the-fourth-one@chat.shakespeare.lit'/>
    <item jid='henry-the-fourth-two@chat.shakespeare.lit'/>
    <item jid='henry-the-fifth@chat.shakespeare.lit'/>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first index='0'>alls-well-that-ends-well@chat.shakespeare.lit</
          first>
      <last>henry-the-fifth@chat.shakespeare.lit</last>
      <count>37</count>
    </set>
  </query>
</iq>
```

6.4 Querying for Room Information

Using the disco#info protocol, an entity may also query a specific chat room for more detailed information about the room. An entity SHOULD do so before entering a room in order to determine the privacy and security profile of the room configuration (see the Security

[9] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[10] XEP-0059: Result Set Management <https://xmpp.org/extensions/xep-0059.html>.

Considerations for details).

Listing 8: Entity Queries for Information about a Specific Chat Room

```
<iq from='hag66@shakespeare.lit/pda'
    id='ik3vs715'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

The room MUST return its identity and SHOULD return the features it supports:

Listing 9: Room Returns Disco Info Result

```
<iq from='coven@chat.shakespeare.lit'
    id='ik3vs715'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='A␣Dark␣Cave'
        type='text'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <feature var='http://jabber.org/protocol/muc#stable_id'/>
    <feature var='muc_passwordprotected'/>
    <feature var='muc_hidden'/>
    <feature var='muc_temporary'/>
    <feature var='muc_open'/>
    <feature var='muc_unmoderated'/>
    <feature var='muc_nonanonymous'/>
  </query>
</iq>
```

Note: The room SHOULD return the materially-relevant features it supports, such as password protection and room moderation (these are listed fully in the feature registry maintained by the XMPP Registrar; see also the XMPP Registrar section of this document).
A chatroom MAY return more detailed information in its disco#info response using Service Discovery Extensions (XEP-0128) [11], identified by inclusion of a hidden FORM_TYPE field whose value is "http://jabber.org/protocol/muc#roominfo". Such information might include a more verbose description of the room, the current room subject, and the current number of occupants in the room:

Listing 10: Room Returns Extended Disco Info Result

[11] XEP-0128: Service Discovery Extensions <https://xmpp.org/extensions/xep-0128.html>.

```
<iq from='coven@chat.shakespeare.lit'
    id='ik3vs715'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='A Dark Cave'
        type='text'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <feature var='muc_passwordprotected'/>
    <feature var='muc_hidden'/>
    <feature var='muc_temporary'/>
    <feature var='muc_open'/>
    <feature var='muc_unmoderated'/>
    <feature var='muc_nonanonymous'/>
    <x xmlns='jabber:x:data' type='result'>
      <field var='FORM_TYPE' type='hidden'>
        <value>http://jabber.org/protocol/muc#roominfo</value>
      </field>
      <field var='muc#roominfo_description'
             label='Description'>
        <value>The place for all good witches!</value>
      </field>
      <field var='muc#roominfo_contactjid'
             label='Contact Addresses'>
        <value>crone1@shakespeare.lit</value>
      </field>
      <field var='muc#roominfo_subject'
             label='Current Discussion Topic'>
        <value>Spells</value>
      </field>
      <field var='muc#roomconfig_changesubject'
             label='Subject can be modified'>
        <value>true</value>
      </field>
      <field var='muc#roominfo_occupants'
             label='Number of occupants'>
        <value>3</value>
      </field>
      <field var='muc#roominfo_ldapgroup'
             label='Associated LDAP Group'>
        <value>cn=witches,dc=shakespeare,dc=lit</value>
      </field>
      <field var='muc#roominfo_lang'
             label='Language of discussion'>
        <value>en</value>
      </field>
      <field var='muc#roominfo_logs'
```

```
            label='URL for discussion logs'>
        <value>http://www.shakespeare.lit/chatlogs/coven/</value>
      </field>
      <field var='muc#maxhistoryfetch'
             label='Maximum Number of History Messages Returned by
                Room'>
        <value>50</value>
      </field>
      <field var='muc#roominfo_pubsub'
             label='Associated pubsub node'>
        <value>xmpp:pubsub.shakespeare.lit?;node=the-coven-node</value
           >
      </field>
    </x>
  </query>
</iq>
```

Some extended room information is dynamically generated (e.g., the URL for discussion logs, which may be based on service-wide configuration), whereas other information is based on the more-stable room configuration, which is why any field defined for the muc#roomconfig FORM_TYPE can be included in the extended service discovery fields (as shown above for the "muc#roomconfig_changesubject" field).

Note: The foregoing extended service discovery fields for the 'http://jabber.org/protocol/muc#roominfo' FORM_TYPE are examples only and might be supplemented in the future via the mechanisms described in the Field Standardization section of this document.

6.5 Querying for Room Items

An entity MAY also query a specific chat room for its associated items:

Listing 11: Entity Queries for Items Associated with a Specific Chat Room

```
<iq from='hag66@shakespeare.lit/pda'
    id='kl2fax27'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

An implementation MAY return a list of existing occupants if that information is publicly available, or return no list at all if this information is kept private. Implementations and deployments are advised to turn off such information sharing by default.

Listing 12: Room Returns Disco Items Result (Items are Public)

```
<iq from='coven@chat.shakespeare.lit'
    id='kl2fax27'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='coven@chat.shakespeare.lit/firstwitch'/>
    <item jid='coven@chat.shakespeare.lit/secondwitch'/>
  </query>
</iq>
```

Note: These <item/> elements are qualified by the disco#items namespace, not the muc namespace; this means that they cannot possess 'affiliation' or 'role' attributes, for example. If the list of occupants is private, the room MUST return an empty <query/> element, in accordance with Service Discovery (XEP-0030) [12].

Listing 13: Room Returns Empty Disco Items Result (Items are Private)

```
<iq from='coven@chat.shakespeare.lit'
    id='kl2fax27'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

6.6 Querying a Room Occupant

If a non-occupant attempts to send a disco request to an address of the form <room@service/nick>, a MUC service MUST return a <bad-request/> error. If an occupant sends such a request, the service MAY pass it through the intended recipient; see the Implementation Guidelines section of this document for details.

6.7 Discovering Client Support for MUC

An entity might want to discover if one of the entity's contacts supports the Multi-User Chat protocol (e.g., before attempting to invite the contact to a room). This can be done using Service Discovery.

Listing 14: Entity Queries Contact Regarding MUC Support

```
<iq from='hag66@shakespeare.lit/pda'
    id='yh2fs843'
    to='wiccarocks@shakespeare.lit/laptop'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
```

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

```
</iq>
```

The client SHOULD return its identity and the features it supports.

Listing 15: Contact Returns Disco Info Result

```
<iq from='wiccarocks@shakespeare.lit/laptop'
    id='yh2fs843'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='client'
        type='pc'/>
    ...
    <feature var='http://jabber.org/protocol/muc'/>
    ...
  </query>
</iq>
```

An entity may also query a contact regarding which rooms the contact is in. This is done by querying the contact's full JID (<user@host/resource>) while specifying the well-known Service Discovery node 'http://jabber.org/protocol/muc#rooms'.

Listing 16: Entity Queries Contact for Current Rooms

```
<iq from='hag66@shakespeare.lit/pda'
    id='gp7w61v3'
    to='wiccarocks@shakespeare.lit/laptop'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://jabber.org/protocol/muc#rooms'/>
</iq>
```

Listing 17: Contact Returns Room Query Result

```
<iq from='wiccarocks@shakespeare.lit/laptop'
    id='gp7w61v3'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://jabber.org/protocol/muc#rooms'>
    <item jid='coven@chat.shakespeare.lit'/>
    <item jid='characters@conference.shakespeare.lit'/>
  </query>
</iq>
```

Optionally, the contact MAY include its roomnick as the value of the 'name' attribute:

```
...
    <item jid='coven@chat.shakespeare.lit'
          name='secondwitch'/>
    ...
```

If this information is private, the user MUST return an empty <query/> element, in accordance with Service Discovery (XEP-0030) [13].

7 Occupant Use Cases

The main actor in a multi-user chat environment is the occupant, who can be said to be located "in" a multi-user chat room and to participate in the discussions held in that room (for the purposes of this specification, participants and visitors are considered to be "mere" occupants, since they possess no admin status). As will become clear, the protocol elements proposed in this document to fulfill the occupant use cases fall into three categories:

1. the basic functionality for joining a room, exchanging messages with all occupants, etc. (supported by the groupchat 1.0 protocol that preceded MUC)

2. straightforward additions to the basic functionality, such as handling of errors related to new room types

3. additional protocol elements to handle functionality not covered by groupchat 1.0 (room invites, room passwords, extended presence related to room roles and affiliations); these are qualified by the 'http://jabber.org/protocol/muc#user' namespace

Note: All client-generated examples herein are presented from the perspective of the service, with the result that all stanzas received by a service contain a 'from' attribute corresponding to the sender's full JID as added by a normal XMPP router or session manager. In addition, normal IQ result stanzas sent upon successful completion of a request (as required by XMPP Core [14]) are not shown.

7.1 Order of Events

The order of events involved in joining a room needs to be consistent so that clients can know which events to expect when. After a client sends presence to join a room, the MUC service MUST send it events in the following order:

[13] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[14] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

1. In-room presence from other occupants
2. In-room presence from the joining entity itself (so-called ”self-presence”)
3. Room history (if any)
4. The room subject
5. Live messages, presence updates, new user joins, etc.

7.2 Entering a Room

7.2.1 Basic MUC Protocol

In order to participate in the discussions held in a multi-user chat room, a user MUST first become an occupant by entering the room.
MUC clients MUST signal their ability to speak the MUC protocol by including in the initial presence stanza an empty <x/> element qualified by the 'http://jabber.org/protocol/muc' namespace (note the absence of the '#user' fragment):

Listing 18: User Seeks to Enter a Room (Multi-User Chat)

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='n13mt3l'
    to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>
```

In this example, a user with a full JID of ”hag66@shakespeare.lit/pda” has requested to enter the room ”coven” on the ”chat.shakespeare.lit” chat service with a room nickname of ”thirdwitch”.
Note: The presence stanza used to join a room MUST NOT possess a 'type' attribute, i.e., it must be available presence. For further discussion, see the Presence business rules.
If the user does not specify a room nickname (note the bare JID on the 'from' address in the following example), the service MUST return a <jid-malformed/> error:

Listing 19: No Nickname Specified

```
<presence
    from='coven@chat.shakespeare.lit'
    id='273hs51g'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error by='coven@chat.shakespeare.lit' type='modify'>
    <jid-malformed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

Before attempting to enter the room, a MUC-compliant client SHOULD first discover its reserved room nickname (if any) by following the protocol defined in the Discovering Reserved Room Nickname section of this document.
When a MUC service receives an <x/> tagged join stanza from an already-joined client (as identified by the client's full JID), the service should assume that the client lost its synchronization, and therefore it SHOULD send exactly the same stanzas to the client as if it actually just joined the MUC. The server MAY also send a presence update to the other participants according to the received join presence.

7.2.2 Presence Broadcast

If the service is able to add the user to the room, it MUST send presence from all the existing participants' occupant JIDs to the new occupant's full JID, including extended presence information about roles in a single <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "moderator", "participant", or "visitor", and with the 'affiliation' attribute set to a value of "owner", "admin", "member", or "none" as appropriate. [15]

Listing 20: Service Sends Presence from Existing Occupants to New Occupant

```
<presence
    from='coven@chat.shakespeare.lit/firstwitch'
    id='3DCB0401-D7CF-4E31-BE05-EDF8D057BFBD'
    to='hag66@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='owner' role='moderator'/>
  </x>
</presence>

<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    id='C2CD9EE3-8421-431E-854A-A2AD0CE2E23D'
    to='hag66@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='admin' role='moderator'/>
  </x>
</presence>
```

In this example, the user from the previous example has entered the room, by which time two other people had already entered the room: a user with a room nickname of "firstwitch" (who is a room owner) and a user with a room nickname of "secondwitch" (who is a room admin). Unless the room is configured to not broadcast presence from new occupants below a certain

[15] The <presence/> element MUST NOT include more than once instance of the <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace.

affiliation level (as controlled by the ”muc#roomconfig_presencebroadcast” room configuration option), the service MUST also send presence from the new participant’s occupant JID to the full JIDs of all the occupants (including the new occupant).

Listing 21: Service Sends New Occupant’s Presence to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='27C55F89-1C6A-459A-9EB5-77690145D624'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member' role='participant'/>
  </x>
</presence>

<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='9E757BAE-8AC8-4093-AA9C-407F6AEF15D6'
    to='wiccarocks@shakespeare.lit/laptop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member' role='participant'/>
  </x>
</presence>

<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='026B3509-2CCE-4D69-96D6-25F41FFDC408'
    to='hag66@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member' role='participant'/>
    <status code='110'/>
  </x>
</presence>
```

In this example, initial room presence is being sent from the new occupant (thirdwitch) to all occupants, including the new occupant.
As shown in the last stanza, the ”self-presence” sent by the room to the new user MUST include a status code of 110 so that the user knows this presence refers to itself as an occupant.
This self-presence MUST NOT be sent to the new occupant until the room has sent the presence of all other occupants to the new occupant; this enables the new occupant to know when it has finished receiving the room roster.
The service MAY rewrite the new occupant’s roomnick (e.g., if roomnicks are locked down or based on some other policy).
The service MUST allow the client to enter the room, modify the nick in accordance with the lockdown policy, and include a status code of ”210” in the presence broadcast that it sends to the new occupant.

Listing 22: Service Sends New Occupant's Presence to New Occupant

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member' role='participant'/>
    <status code='110'/>
    <status code='210'/>
  </x>
</presence>
```

Note: The order of the presence stanzas sent to the new occupant is important. The service MUST first send the complete list of the existing occupants to the new occupant and only then send the new occupant's own presence to the new occupant. This helps the client know when it has received the complete "room roster". For tracking purposes, the room might also reflect the original 'id' value if provided in the presence stanza sent by the user.
After sending the presence broadcast (and only after doing so), the service can then send discussion history, the room subject, live messages, presence updates, and other in-room traffic.

7.2.3 Non-Anonymous Rooms

If the room is non-anonymous, the service MUST send the new occupant's full JID to all occupants using extended presence information in an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with a 'jid' attribute specifying the occupant's full JID:

Listing 23: Service Sends Full JID to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='17232D15-134F-43C8-9A29-61C20A64B236'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
  </x>
</presence>

[ ... ]
```

If the user is entering a room that is non-anonymous (i.e., which informs all occupants of each occupant's full JID as shown above), the service MUST warn the user by including a status code of "100" in the initial presence that the room sends to the new occupant:

Listing 24: Service Sends New Occupant's Presence to New Occupant

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member' role='participant'/>
    <status code='100'/>
    <status code='110'/>
    <status code='210'/>
  </x>
</presence>
```

The inclusion of the status code assists clients in presenting their own notification messages (e.g., information appropriate to the user's locality).

7.2.4 Semi-Anonymous Rooms

If the room is semi-anonymous, the service MUST send presence from the new occupant to all occupants as specified above (i.e., unless the room is configured to not broadcast presence from new occupants below a certain affiliation level as controlled by the "muc#roomconfig_-presencebroadcast" room configuration option), but MUST include the new occupant's full JID only in the presence notifications it sends to occupants with a role of "moderator" and not to non-moderator occupants.
(Note: All subsequent examples include the 'jid' attribute for each <item/> element, even though this information is not sent to non-moderators in semi-anonymous rooms.)

7.2.5 Password-Protected Rooms

If the room requires a password and the user did not supply one (or the password provided is incorrect), the service MUST deny access to the room and inform the user that they are unauthorized; this is done by returning a presence stanza of type "error" specifying a <not-authorized/> error:

Listing 25: Service Denies Access Because No Password Provided

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='auth'>
```

```
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

Passwords SHOULD be supplied with the presence stanza sent when entering the room, contained within an <x/> element qualified by the 'http://jabber.org/protocol/muc' namespace and containing a <password/> child. Passwords are to be sent as cleartext; no other authentication methods are supported at this time, and any such authentication or authorization methods shall be defined in a separate specification (see the Security Considerations section of this document).

Listing 26: User Provides Password On Entering a Room

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='djn4714'
    to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'>
    <password>cauldronburn</password>
  </x>
</presence>
```

7.2.6 Members-Only Rooms

If the room is members-only but the user is not on the member list, the service MUST deny access to the room and inform the user that they are not allowed to enter the room; this is done by returning a presence stanza of type "error" specifying a <registration-required/> error condition:

Listing 27: Service Denies Access Because User Is Not on Member List

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='auth'>
    <registration-required xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'
      />
  </error>
</presence>
```

7.2.7 Banned Users

If the user has been banned from the room (i.e., has an affiliation of "outcast"), the service MUST deny access to the room and inform the user of the fact that they are banned; this is

done by returning a presence stanza of type "error" specifying a <forbidden/> error condition:

Listing 28: Service Denies Access Because User is Banned

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

7.2.8 Nickname Conflict

If the room already contains another user with the nickname desired by the user seeking to enter the room (or if the nickname is reserved by another user on the member list), the service MUST deny access to the room and inform the user of the conflict; this is done by returning a presence stanza of type "error" specifying a <conflict/> error condition:

Listing 29: Service Denies Access Because of Nick Conflict

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

However, if the bare JID <localpart@domain.tld> of the present occupant matches the bare JID of the user seeking to enter the room, then the service SHOULD allow entry to the user, so that the user has two (or more) in-room "sessions" with the same roomnick, one for each resource. If a service allows more than one occupant with the same bare JID and the same room nickname, it MUST route in-room messages to all of the user's resources and allow all of the user's resources to send messages to the room; it is up to the implementation whether to route private messages to all resources or only one resource (based on presence priority or some other algorithm); however, it is RECOMMENDED to route to all resources.
How nickname conflicts are determined is up to the implementation (e.g., whether the service applies a case folding routine, a stringprep profile such as Resourceprep or Nodeprep, etc.).

7.2.9 Max Users

If the room has reached its maximum number of occupants, the service SHOULD deny access to the room and inform the user of the restriction; this is done by returning a presence stanza of type "error" specifying a <service-unavailable/> error condition:

Listing 30: Service Informs User that Room Occupant Limit Has Been Reached

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='wait'>
    <service-unavailable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

Alternatively, the room could kick an "idle user" in order to free up space (where the definition of "idle user" is up to the implementation).
If the room has reached its maximum number of occupants and a room admin or owner attempts to join, the room MUST allow the admin or owner to join, up to some reasonable number of additional occupants; this helps to prevent denial of service attacks caused by stuffing the room with non-admin users.

7.2.10 Locked Room

If a user attempts to enter a room while it is "locked" (i.e., before the room creator provides an initial configuration and therefore before the room officially exists), the service MUST refuse entry and return an <item-not-found/> error to the user:

Listing 31: Service Denies Access Because Room Does Not (Yet) Exist

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

7.2.11 Nonexistent Room

If the room does not already exist when the user seeks to enter it, the service SHOULD create it; however, this is not required, since an implementation or deployment MAY choose to restrict the privilege of creating rooms. For details, see the Creating a Room section of this document.

7.2.12 Room Logging

If the user is entering a room in which the discussions are logged to a public archive (often accessible via HTTP), the service SHOULD allow the user to enter the room but MUST also warn the user that the discussions are logged. This is done by including a status code of "170" in the initial presence that the room sends to the new occupant:

Listing 32: Service Sends New Occupant's Presence to New Occupant

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='n13mt3l'
    to='hag66@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member' role='participant'/>
    <status code='100'/>
    <status code='110'/>
    <status code='170'/>
    <status code='210'/>
  </x>
</presence>
```

7.2.13 Discussion History

After sending initial presence as shown above, depending on local service policy or room configuration a room MAY send discussion history to the new occupant. (The room MUST NOT send any discussion history before it finishes sending room presence as specified in the Presence Broadcast section of this document.) Whether such history is sent, and how many messages comprise the history, shall be determined by the chat service implementation or specific deployment depending on local service policy or room configuration.

Listing 33: Delivery of Discussion History

```
<message
    from='coven@chat.shakespeare.lit/firstwitch'
    id='162BEBB1-F6DB-4D9A-9BD8-CFDCC801A0B2'
    to='hecate@shakespeare.lit/broom'
    type='groupchat'>
```

```
  <body>Thrice the brinded cat hath mew'd.</body>
  <delay xmlns='urn:xmpp:delay'
      from='coven@chat.shakespeare.lit'
      stamp='2002-10-13T23:58:37Z'/>
</message>

<message
    from='coven@chat.shakespeare.lit/secondwitch'
    id='90057840-30FD-4141-AA44-103EEDF218FC'
    to='hecate@shakespeare.lit/broom'
    type='groupchat'>
  <body>Thrice and once the hedge-pig whined.</body>
  <delay xmlns='urn:xmpp:delay'
      from='coven@chat.shakespeare.lit'
      stamp='2002-10-13T23:58:43Z'/>
</message>

<message
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
    to='hecate@shakespeare.lit/broom'
    type='groupchat'>
  <body>Harpier cries 'Tis time, 'tis time.</body>
  <delay xmlns='urn:xmpp:delay'
      from='coven@chat.shakespeare.lit'
      stamp='2002-10-13T23:58:49Z'/>
</message>
```

Discussion history messages MUST be stamped with Delayed Delivery (XEP-0203) [16] information qualified by the 'urn:xmpp:delay' namespace to indicate that they are sent with delayed delivery and to specify the times at which they were originally sent. The 'from' attribute MUST be set to the JID of the room itself.
(Note: The 'urn:xmpp:delay' namespace defined in Delayed Delivery (XEP-0203) [17] supersedes the older 'jabber:x:delay' namespace defined in Legacy Delayed Delivery (XEP-0091) [18]; some implementations include both formats for backward compatibility.)
The service MUST send all discussion history messages before delivering the room subject and any "live" messages sent after the user enters the room. Note well that this means the room subject (and changes to the room subject prior to the current subject) are not part of the discussion history.
If the room is non-anonymous, the service MAY include an Extended Stanza Addressing (XEP-0033) [19] element that notes the original full JID of the sender by means of the "ofrom" address type:

[16] XEP-0203: Delayed Delivery <https://xmpp.org/extensions/xep-0203.html>.
[17] XEP-0203: Delayed Delivery <https://xmpp.org/extensions/xep-0203.html>.
[18] XEP-0091: Legacy Delayed Delivery <https://xmpp.org/extensions/xep-0091.html>.
[19] XEP-0033: Extended Stanza Addressing <https://xmpp.org/extensions/xep-0033.html>.

Listing 34: Discussion History Message with Original From

```
<message
    from='coven@chat.shakespeare.lit/firstwitch'
    id='162BEBB1-F6DB-4D9A-9BD8-CFDCC801A0B2'
    to='hecate@shakespeare.lit/broom'
    type='groupchat'>
  <body>Thrice the brinded cat hath mew'd.</body>
␣␣<delay␣xmlns='urn:xmpp:delay'
␣␣␣␣␣from='coven@chat.shakespeare.lit'
␣␣␣␣␣stamp='2002-10-13T23:58:37Z'/>
␣␣<addresses␣xmlns='http://jabber.org/protocol/address'>
␣␣␣␣<address␣type='ofrom'␣jid='crone1@shakespeare.lit/desktop'/>
␣␣</addresses>
</message>
```

7.2.14 Managing Discussion History

A user might want to manage the amount of discussion history provided on entering a room (perhaps because the user is on a low-bandwidth connection or is using a small-footprint client). This is accomplished by including a <history/> child in the initial presence stanza sent when joining the room. There are four allowable attributes for this element:

Attribute	Datatype	Meaning
maxchars	int	Limit the total number of characters in the history to "X" (where the character count is the characters of the complete XML stanzas, not only their XML character data).
maxstanzas	int	Limit the total number of messages in the history to "X".
seconds	int	Send only the messages received in the last "X" seconds.
since	dateTime	Send only the messages received since the UTC datetime specified (which MUST conform to the DateTime profile specified in XMPP Date and Time Profiles (XEP-0082) XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>.).

The service MUST send the smallest amount of traffic that meets any combination of the above criteria, taking into account service-level and room-level defaults. The service MUST

send complete message stanzas only (i.e., it MUST not literally truncate the history at a certain number of characters, but MUST send the largest number of complete stanzas that results in a number of characters less than or equal to the 'maxchars' value specified). If the client wishes to receive no history, it MUST set the 'maxchars' attribute to a value of "0" (zero).
Note: It is known that not all service implementations support MUC history management, so in practice a client might not be able to depend on receiving only the history that it has requested.
The following examples illustrate the use of this feature.

Listing 35: User Requests Limit on Number of Characters in History

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='n13mt3l'
    to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'>
    <history maxchars='65000'/>
  </x>
</presence>
```

Listing 36: User Requests Limit on Number of Messages in History

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='n13mt3l'
    to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'>
    <history maxstanzas='20'/>
  </x>
</presence>
```

Listing 37: User Requests History in Last 3 Minutes

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='n13mt3l'
    to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'>
    <history seconds='180'/>
  </x>
</presence>
```

Listing 38: User Requests All History Since the Beginning of the Unix Era

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='n13mt3l'
    to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'>
```

```
    <history since='1970-01-01T00:00:00Z'/>
  </x>
</presence>
```

Obviously the service SHOULD NOT return all messages sent in the room since the beginning of the Unix era, and SHOULD appropriately limit the amount of history sent to the user based on service or room defaults.

Listing 39: User Requests No History

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='n13mt3l'
    to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'>
    <history maxchars='0'/>
  </x>
</presence>
```

7.2.15 Room Subject

After the room has optionally sent the discussion history to the new occupant, it SHALL send the current room subject. This is a <message/> stanza from the room JID (or from the occupant JID of the entity that set the subject), with a <subject/> element but no <body/> element, as shown in the following example. In addition, the subject SHOULD be stamped with Delayed Delivery (XEP-0203) [20] information qualified by the 'urn:xmpp:delay' namespace to indicate the time at which the subject was last modified. If the <delay/> element is included, its 'from' attribute MUST be set to the JID of the room itself.

Listing 40: Service Informs New Occupant of Room Subject

```
<message
    from='coven@chat.shakespeare.lit/secondwitch'
    id='F437C672-D438-4BD3-9BFF-091050D32EE2'
    to='crone1@shakespeare.lit/desktop'
    type='groupchat'>
  <subject>Fire Burn and Cauldron Bubble!</subject>
  <delay xmlns='urn:xmpp:delay' from='coven@chat.shakespeare.lit'
     stamp='1610-04-20T00:00:00Z'/>
</message>
```

Interoperability Note: The <delay/> element has been specified in version 1.34.0 of this document. Hence, consuming entities need to be able to deal with servers which do not send a <delay/> element. Most notably, this means that the presence of the <delay/> element cannot be used to distinguish a historic vs. a live subject change.

[20] XEP-0203: Delayed Delivery <https://xmpp.org/extensions/xep-0203.html>.

If there is no subject set, the room MUST return an empty <subject/> element. The <delay/> SHOULD be included if the subject was actively cleared and MAY be omitted if the room never had a subject set.

Listing 41: No Subject

```
<message
    from='coven@chat.shakespeare.lit/secondwitch'
    id='F437C672-D438-4BD3-9BFF-091050D32EE2'
    to='crone1@shakespeare.lit/desktop'
    type='groupchat'>
  <subject></subject>
</message>
```

Note: In accordance with the core definition of XML stanzas, any message can contain a <subject/> element; only a message that contains a <subject/> but no <body/> element shall be considered a subject change for MUC purposes.

7.2.16 Live Messages

After the room has sent the room subject, it SHALL begin to send live messages, presence changes, occupant "joins" and "leaves", and other real-time traffic to the new occupant, as described in other sections of this document.

7.2.17 Error Conditions

The following table summarizes the XMPP error conditions that can be returned to an entity that attempts to enter a MUC room.

Condition	Purpose
<not-authorized/>	Inform user that a password is required
<forbidden/>	Inform user that he or she is banned from the room
<item-not-found/>	Inform user that the room does not exist
<not-allowed/>	Inform user that room creation is restricted
<not-acceptable/>	Inform user that the reserved roomnick must be used
<registration-required/>	Inform user that he or she is not on the member list
<conflict/>	Inform user that his or her desired room nickname is in use or registered by another user
<service-unavailable/>	Inform user that the maximum number of users has been reached

7.2.18 Groupchat 1.0 Protocol

In the old groupchat 1.0 protocol, entering a room was done by sending presence with no 'type' attribute to <room@service/nick>, where "room" is the room ID, "service" is the hostname of the chat service, and "nick" is the user's desired nickname within the room:

Listing 42: User Seeks to Enter a Room (groupchat 1.0)

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='ng91xs69'
    to='coven@chat.shakespeare.lit/thirdwitch'/>
```

This behavior can not be distinguished from a presence update from a MUC-supporting client that was desynchronized from the room. Treating this as a groupchat 1.0 join will mask the error and leave the client in a partially-synchronized state. Therefore, starting with version 1.32 of this specification, it is RECOMMENDED that a service receiving a <presence> without an <x> element from a non-occupant full-JID responds with an explicit kick to that client. The kick MUST contain the status codes 110 (occupant's presence), 307 (kick), and 333 (kick due to technical problems).

Listing 43: Service Response to groupchat 1.0 join / non-occupant presence update

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='hag66@shakespeare.lit/pda'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none'>
      <reason>You are not in the room.</reason>
    </item>
    <status code='110'/>
    <status code='307'/>
    <status code='333'/>
  </x>
</presence>
```

7.3 Occupant Modification of the Room Subject

If allowed in accordance with room configuration, a mere occupant MAY be allowed to change the subject in a room. For details, see the Modifying the Room Subject section of this document.

7.4 Sending a Message to All Occupants

An occupant sends a message to all other occupants in the room by sending a message of type "groupchat" to the <room@service> itself (a service MAY ignore or reject messages that do

not have a type of "groupchat"). In a moderated room, this privilege is restricted to occupants with a role of participant or higher.

Listing 44: Occupant Sends a Message to All Occupants

```
<message
    from='hag66@shakespeare.lit/pda'
    id='hysf1v37'
    to='coven@chat.shakespeare.lit'
    type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
</message>
```

If the sender has voice in the room (this is the default except in moderated rooms) and the message does not violate any service-level or room-level policies (e.g., policies regarding message content or size), the service MUST change the 'from' attribute to the sender's occupant JID and reflect the message out to the full JID of each occupant.

Listing 45: Service Reflects Message to All Occupants

```
<message
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='hysf1v37'
    to='crone1@shakespeare.lit/desktop'
    type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
</message>
<message
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='hysf1v37'
    to='wiccarocks@shakespeare.lit/laptop'
    type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
</message>
<message
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='hysf1v37'
    to='hag66@shakespeare.lit/pda'
    type='groupchat'>
  <body>Harpier cries: 'tis␣time,␣'tis time.</body>
</message>
```

The service SHOULD reflect the message with the same 'id' that was generated by the client, to allow clients to track their outbound messages. If the client did not provide an 'id', the server MAY generate an 'id' and use it for all reflections of the same message (e.g. using a UUID as defined in RFC 4122 [21]).

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

Note: the requirement to reflect the 'id' attribute was added in version 1.31 of this XEP. Servers following the new specification SHOULD advertise that with a disco info feature of 'http://jabber.org/protocol/muc#stable_id' on both the service domain and on individual MUCs, so that clients can check for support.
If the sender is a visitor (i.e., does not have voice in a moderated room), the service MUST return a <forbidden/> error to the sender and MUST NOT reflect the message to all occupants. If the sender is not an occupant of the room, the service SHOULD return a <not-acceptable/> error to the sender and SHOULD NOT reflect the message to all occupants; the only exception to this rule is that an implementation MAY allow users with certain privileges (e.g., a room owner, room admin, or service-level admin) to send messages to the room even if those users are not occupants.

7.5 Sending a Private Message

Since each occupant has its own occupant JID, an occupant can send a "private message" to a selected occupant via the service by sending a message to the intended recipient's occupant JID. The message type SHOULD be "chat" and MUST NOT be "groupchat", but MAY be left unspecified (i.e., a normal message). This privilege is controlled by the "muc#roomconfig_allowpm" room configuration option.
To allow for proper synchronization of these messages to the user's other clients by Message Carbons (XEP-0280) [22], the sending client SHOULD add an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace to the message.
Note: because this requirement was only added in revision 1.28 of this XEP, receiving entities MUST NOT rely on the existence of the <x/> element on private messages for proper processing.

Listing 46: Occupant Sends Private Message

```
<message
    from='wiccarocks@shakespeare.lit/laptop'
    id='hgn27af1'
    to='coven@chat.shakespeare.lit/firstwitch'
    type='chat'>
  <body>I'll give thee a wind.</body>
  <x xmlns='http://jabber.org/protocol/muc#user' />
</message>
```

The service is responsible for changing the 'from' address to the sender's occupant JID and delivering the message to the intended recipient's full JID. The service SHOULD add the <x/> element if the message does not contain it already.

Listing 47: Recipient Receives the Private Message

[22] XEP-0280: Message Carbons <https://xmpp.org/extensions/xep-0280.html>.

```
<message
    from='coven@chat.shakespeare.lit/secondwitch'
    id='hgn27af1'
    to='crone1@shakespeare.lit/desktop'
    type='chat'>
  <body>I'll␣give␣thee␣a␣wind.</body>
␣␣<x␣xmlns='http://jabber.org/protocol/muc#user'␣/>
</message>
```

If the sender attempts to send a private message of type "groupchat" to a particular occupant, the service MUST refuse to deliver the message (since the recipient's client would expect in-room messages to be of type "groupchat") and return a <bad-request/> error to the sender:

Listing 48: Occupant Attempts to Send a Message of Type "Groupchat" to a Particular Occupant

```
<message
    from='wiccarocks@shakespeare.lit/laptop'
    id='bx71f29k'
    to='coven@chat.shakespeare.lit/firstwitch'
    type='groupchat'>
  <body>I'll␣give␣thee␣a␣wind.</body>
</message>

<message
␣␣␣␣from='coven@chat.shakespeare.lit/firstwitch'
␣␣␣␣id='bx71f29k'
␣␣␣␣to='wiccarocks@shakespeare.lit/laptop'
␣␣␣␣type='error'>
␣␣<body>I'll give thee a wind.</body>
  <error by='coven@chat.shakespeare.lit' type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</message>
```

If the sender attempts to send a private message to an occupant JID that does not exist, the service MUST return an <item-not-found/> error to the sender.
If the sender is not an occupant of the room in which the intended recipient is visiting, the service MUST return a <not-acceptable/> error to the sender.

7.6 Changing Nickname

A common feature of chat rooms is the ability for an occupant to change his or her nickname within the room. In MUC this is done by sending updated presence information to the room, specifically by sending presence to a new occupant JID in the same room (changing only the resource identifier in the occupant JID).

Listing 49: Occupant Changes Nickname

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='ifd1c35'
    to='coven@chat.shakespeare.lit/oldhag'/>
```

The service then sends two presence stanzas to the full JID of each occupant (including the occupant who is changing his or her room nickname), one of type "unavailable" for the old nickname and one indicating availability for the new nickname.
The unavailable presence MUST contain the following as extended presence information in an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace:

- The new nickname (in this case, nick='oldhag')
- A status code of 303

This enables the recipients to correlate the old roomnick with the new roomnick.

Listing 50: Service Updates Nick

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='5C1B95B3-7CCC-4422-A952-8885A050BDE9'
    to='crone1@shakespeare.lit/desktop'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          nick='oldhag'
          role='participant'/>
    <status code='303'/>
  </x>
</presence>
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='B0E6ABD5-575D-42F0-8242-569004D88F73'
    to='wiccarocks@shakespeare.lit/laptop'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          nick='oldhag'
          role='participant'/>
    <status code='303'/>
  </x>
</presence>
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
```

```
    id='DC352437-C019-40EC-B590-AF29E879AF98'
    to='hag66@shakespeare.lit/pda'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          nick='oldhag'
          role='participant'/>
    <status code='303'/>
    <status code='110'/>
  </x>
</presence>

<presence
    from='coven@chat.shakespeare.lit/oldhag'
    id='19E41EB3-3F4C-444F-8A1B-713A8860980C'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
  </x>
</presence>
<presence
    from='coven@chat.shakespeare.lit/oldhag'
    id='79225CEA-F610-49BE-9B97-FEFA8737185B'
    to='wiccarocks@shakespeare.lit/laptop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
  </x>
</presence>
<presence
    from='coven@chat.shakespeare.lit/oldhag'
    id='5B4F27A4-25ED-43F7-A699-382C6B4AFC67'
    to='hag66@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
    <status code='110'/>
  </x>
</presence>
```

If the service modifies the user's nickname in accordance with local service policies, it MUST include a MUC status code of 210 in the presence stanza sent to the user. An example follows (here the service changes the nickname to all lowercase).

Listing 51: Occupant Changes Nickname, Modified by Service

```
<presence
    from='hag66@shakespeare.lit/pda'
    id='nx6z2v5'
    to='coven@chat.shakespeare.lit/OldHag'/>

<presence
    from='coven@chat.shakespeare.lit/oldhag'
    id='D0E2B666-3373-42C9-B726-D52C40A48383'
    to='hag66@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
    <status code='110'/>
    <status code='210'/>
  </x>
</presence>
```

If the user attempts to change his or her room nickname to a room nickname that is already in use by another user (or that is reserved by another user affiliated with the room, e.g., a member or owner), the service MUST deny the nickname change request and inform the user of the conflict; this is done by returning a presence stanza of type "error" specifying a <conflict/> error condition:

Listing 52: Service Denies Nickname Change Because of Nick Conflict

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='ifd1c35'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

However, if the bare JID <localpart@domain.tld> of the present occupant matches the bare JID of the user seeking to change his or her nickname, then the service MAY allow the nickname change. See the Nickname Conflict section of this document for details.
If the user attempts to change their room nickname but nicknames are "locked down", the service MUST either deny the nickname change request and return a presence stanza of type "error" with a <not-acceptable/> error condition:

Listing 53: Service Denies Nickname Change Because Roomnicks Are Locked Down

```
<presence
```

```
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='ifd1c35'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='cancel'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

The user SHOULD then discover its reserved nickname as specified in the Discovering Reserved Room Nickname section of this document.

7.7 Changing Availability Status

In text chat systems such as IRC, one common use for changing one's room nickname is to indicate a change in one's availability (e.g., changing one's room nickname to "third-witch|away"). In XMPP, availability is of course noted by a change in presence (specifically the <show/> and <status/> elements), which can provide important context within a chatroom. An occupant changes availability status within the room by sending updated presence to its <room@service/nick>.

Listing 54: Occupant Changes Availability Status

```
<presence
    from='wiccarocks@shakespeare.lit/laptop'
    id='kr7v143h'
    to='coven@chat.shakespeare.lit/oldhag'>
  <show>xa</show>
  <status>gone where the goblins go</status>
</presence>
```

If the room is configured to broadcast presence from entities with the occupant's role, the service then sends a presence stanza from the occupant changing his or her presence to the full JID of each occupant, including extended presence information about the occupant's role and full JID to those with privileges to view such information:

Listing 55: Service Passes Along Changed Presence to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    id='86E11ABF-26BC-46F1-AD3B-F5E54F3C1EE5'
    to='crone1@shakespeare.lit/desktop'>
  <show>xa</show>
  <status>gone where the goblins go</status>
  <x xmlns='http://jabber.org/protocol/muc#user'>
```

```
    <item affiliation='admin'
          jid='wiccarocks@shakespeare.lit/laptop'
          role='moderator'/>
  </x>
</presence>

[ ... ]
```

7.8 Inviting Another User to a Room

There are two ways of inviting another user to a room: direct invitations and mediated invitations.
Direct invitations were the original method used in the early Jabber community's "groupchat 1.0" protocol. Mediated invitations were added in Multi-User Chat as a way to handle invitations in the context of members-only rooms (so that the room could exercise control over the issuance of invitations). The existence of two different invitation methods might cause confusion among client developers. Because the room needs to be involved in the invitation process only for members-only rooms, because members-only rooms are relatively rare, and because mediated invitations do not work when Privacy Lists (XEP-0016) [23] or similar technologies are used to block communication from entities not in a user's roster, client developers are encouraged to use direct invitations for all other room types.

7.8.1 Direct Invitation

A method for sending a direct invitation (not mediated by the room itself) is defined in Direct MUC Invitations (XEP-0249) [24]. Sending the invitation directly can help to work around communications blocking on the part of the invitee (which might reject or discard messages from entities not in its roster).

7.8.2 Mediated Invitation

It can be useful to invite another user to a room in which one is an occupant. To send a mediated invitation, a MUC client MUST send XML of the following form to the <room@service> itself (the reason is OPTIONAL and the message MUST be explicitly or implicitly of type "normal"):

Listing 56: Occupant Sends a Mediated Invitation

```
<message
    from='crone1@shakespeare.lit/desktop'
    id='nzd143v8'
```

[23] XEP-0016: Privacy Lists <https://xmpp.org/extensions/xep-0016.html>.
[24] XEP-0249: Direct MUC Invitations <https://xmpp.org/extensions/xep-0249.html>.

```
    to='coven@chat.shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite to='hecate@shakespeare.lit'>
      <reason>
        Hey Hecate, this is the place for all good witches!
      </reason>
    </invite>
  </x>
</message>
```

The <room@service> itself MUST then add a 'from' address to the <invite/> element whose value is the bare JID, full JID, or occupant JID of the inviter and send the invitation to the invitee specified in the 'to' address; the room SHOULD add the password if the room is password-protected):

Listing 57: Room Sends Invitation to Invitee on Behalf of Invitor

```
<message
    from='coven@chat.shakespeare.lit'
    id='nzd143v8'
    to='hecate@shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite from='crone1@shakespeare.lit/desktop'>
      <reason>
        Hey Hecate, this is the place for all good witches!
      </reason>
    </invite>
    <password>cauldronburn</password>
  </x>
</message>
```

If the room is members-only, the service MAY also add the invitee to the member list. (Note: Invitation privileges in members-only rooms SHOULD be restricted to room admins; if a member without privileges to edit the member list attempts to invite another user, the service SHOULD return a <forbidden/> error to the occupant; for details, see the Modifying the Member List section of this document.)

Implementation Note: In the past, it was specified that a <x xmlns='jabber:x:conference'> element with the reason as text payload was to be included in the mediated invitation as sent by the room. While this has since been removed from this specification, implementations should be aware that there still exist server implementations which emit that payload for compatibility reasons.

If the inviter supplies a non-existent JID, the room SHOULD return an <item-not-found/> error to the inviter.

The invitee MAY choose to formally decline (as opposed to ignore) the invitation; and this is something that the sender might want to be informed about. In order to decline the invitation, the invitee MUST send a message of the following form to the <room@service> itself:

Listing 58: Invitee Declines Invitation

```
<message
    from='hecate@shakespeare.lit/broom'
    id='jk2vs61v'
    to='coven@chat.shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <decline to='crone1@shakespeare.lit'>
      <reason>
        Sorry, I'm too busy right now.
      </reason>
    </decline>
  </x>
</message>
```

Listing 59: Room Informs Invitor that Invitation Was Declined

```
<message
    from='coven@chat.shakespeare.lit'
    id='jk2vs61v'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <decline from='hecate@shakespeare.lit'>
      <reason>
        Sorry, I'm too busy right now.
      </reason>
    </decline>
  </x>
</message>
```

It may be wondered why the invitee does not send the decline message directly to the inviter. The main reason is that certain implementations might choose to base invitations on occupant JIDs rather than bare JIDs (so that, for example, an occupant could invite someone from one room to another without knowing that person's bare JID). Thus the service needs to handle both the invites and declines.

7.9 Converting a One-to-One Chat Into a Multi-User Conference

Sometimes it is desirable to convert a one-to-one chat into a multi-user conference. The process is as follows.
First, two users begin a one-to-one chat.

Listing 60: A One-to-One Chat

```
<message
    from='crone1@shakespeare.lit/desktop'
```

```
    id='mjs51f36'
    to='wiccarocks@shakespeare.lit/laptop'
    type='chat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <body>Thrice the brinded cat hath mew'd.</body>
</message>

<message
    from='wiccarocks@shakespeare.lit/laptop'
    id='l9ij1f3h'
    to='crone1@shakespeare.lit/desktop'
    type='chat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <body>Thrice and once the hedge-pig whined.</body>
</message>
```

Now the first person decides to include a third person in the discussion, so she does the following:

1. Creates a new multi-user chatroom
2. Sends history of the one-to-one chat to the room (this is purely discretionary; however, because it might cause information leakage, the client ought to warn the user before doing so)
3. Sends an invitation to the second person and the third person, including a <continue/> element (optionally including a 'thread' attribute).

Note: The new room SHOULD be non-anonymous and MAY be an instant room as specified in the Creating an Instant Room section of this document.
Note: If the one-to-one chat messages included a <thread/> element, the person who creates the room SHOULD include the ThreadID with the history messages, specify the ThreadID in the invitations as the value of the <continue/> element's 'thread' attribute, and include the ThreadID in any new messages sent to the room. Use of ThreadIDs is RECOMMENDED because it helps to provide continuity between the one-to-one chat and the multi-user chat.

Listing 61: Continuing the Discussion I: User Creates Room

```
<presence
    from='crone1@shakespeare.lit/desktop'
    to='coven@chat.shakespeare.lit/firstwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>

<presence
    from='coven@chat.shakespeare.lit/firstwitch'
    to='crone1@shakespeare.lit/desktop'>
```

```
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='owner' role='moderator'/>
    <status code='110'/>
  </x>
</presence>
```

Listing 62: Continuing the Discussion II: Owner Sends History to Room

```
<message
    from='crone1@shakespeare.lit/desktop'
    id='b4va73n0'
    to='coven@chat.shakespeare.lit'
    type='groupchat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <body>Thrice the brinded cat hath mew'd.</body>
  <delay xmlns='urn:xmpp:delay'
      from='crone1@shakespeare.lit/desktop'
      stamp='2004-09-29T01:54:37Z'/>
</message>

<message
    from='crone1@shakespeare.lit/desktop'
    id='i4hs759k'
    to='coven@chat.shakespeare.lit'
    type='groupchat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <body>Thrice and once the hedge-pig whined.</body>
  <delay xmlns='urn:xmpp:delay'
      from='crone1@shakespeare.lit/desktop'
      stamp='2004-09-29T01:55:21Z'/>
</message>
```

Note: Use of the Delayed Delivery protocol enables the room creator to specify the datetime of each message from the one-to-one chat history (via the 'stamp' attribute), as well as the JID of the original sender of each message (via the 'from' attribute); note well that the 'from' here is not the room itself, since the originator of the message is the delaying party. The room creator might send the complete one-to-one chat history before inviting additional users to the room, and also send as history any messages appearing in the one-to-one chat interface after joining the room and before the second person joins the room; if the one-to-one history is especially large, the sending client might want to send the history over a few seconds rather than all at once (to avoid triggering rate limits). The service SHOULD NOT add its own delay elements (as described in the Discussion History section of this document) to prior chat history messages received from the room owner.

Listing 63: Continuing the Discussion III: Owner Sends Invitations, Including Continue Flag

```
<message
    from='crone1@shakespeare.lit/desktop'
```

```
    id='gl3s85n7'
    to='coven@chat.shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite to='wiccarocks@shakespeare.lit/laptop'>
      <reason>This coven needs both wiccarocks and hag66.</reason>
      <continue thread='e0ffe42b28561960c6b12b944a092794b9683a38'/>
    </invite>
    <invite to='hag66@shakespeare.lit'>
      <reason>This coven needs both wiccarocks and hag66.</reason>
      <continue thread='e0ffe42b28561960c6b12b944a092794b9683a38'/>
    </invite>
  </x>
</message>
```

Note: Since the inviter's client knows the full JID of the person with whom the inviter was having a one-to-one chat, it SHOULD include the full JID (rather than the bare JID) in its invitation to that user.
The invitations are delivered to the invitees:

Listing 64: Invitations Delivered

```
<message
    from='coven@chat.shakespeare.lit/firstwitch'
    id='DB0414CB-AFBA-407E-9DE3-0E014E84860F'
    to='wiccarocks@shakespeare.lit/laptop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite from='crone1@shakespeare.lit'>
      <reason>This coven needs both wiccarocks and hag66.</reason>
      <continue thread='e0ffe42b28561960c6b12b944a092794b9683a38'/>
    </invite>
  </x>
</message>

<message
    from='coven@chat.shakespeare.lit/firstwitch'
    id='89028D79-AB4C-44C0-BE81-B07607C2F4C2'
    to='hag66@shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite from='crone1@shakespeare.lit'>
      <reason>This coven needs both wiccarocks and hag66.</reason>
      <continue thread='e0ffe42b28561960c6b12b944a092794b9683a38'/>
    </invite>
  </x>
</message>
```

When the client being used by <wiccarocks@shakespeare.lit/laptop> receives the invitation, it can either auto-join the room or prompt the user whether to join (subject to user preferences) and then seamlessly convert the existing one-to-one chat window into a multi-user conferencing window:

Listing 65: Invitee Accepts Invitation, Joins Room, and Receives Presence and History

```
<presence
    from='wiccarocks@shakespeare.lit/laptop'
    to='coven@chat.shakespeare.lit/secondwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>

<presence
    from='coven@chat.shakespeare.lit/firstwitch'
    to='wiccarocks@shakespeare.lit/laptop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='owner' role='moderator'/>
  </x>
</presence>

<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    to='wiccarocks@shakespeare.lit/laptop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member' role='participant'/>
  </x>
</presence>

<message
    from='coven@chat.shakespeare.lit'
    id='67268D36-100C-457D-A769-8A3663BD1949'
    to='wiccarocks@shakespeare.lit/laptop'
    type='groupchat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <body>Thrice the brinded cat hath mew'd.</body>
␣␣<delay␣xmlns='urn:xmpp:delay'
␣␣␣␣␣from='coven@chat.shakespeare.lit'
␣␣␣␣␣stamp='2004-09-29T01:54:37Z'/>
</message>

<message
␣␣␣␣from='coven@chat.shakespeare.lit'
␣␣␣␣id='367DCF6B-0CB4-482D-A142-C0B9E08016B5'
␣␣␣␣to='wiccarocks@shakespeare.lit/laptop'
␣␣␣␣type='groupchat'>
␣␣<thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
␣␣<body>Thrice␣and␣once␣the␣hedge-pig␣whined.</body>
␣␣<delay␣xmlns='urn:xmpp:delay'
␣␣␣␣␣from='coven@chat.shakespeare.lit'
␣␣␣␣␣stamp='2004-09-29T01:55:21Z'/>
</message>
```

7.10 Registering with a Room

An implementation MAY allow an unaffiliated user (in a moderated room, normally a participant) to register with a room; as a result, the user will become a member of the room and will have their preferred nickname reserved in the room. (Conversely, an implementation MAY restrict this privilege and allow only room admins to add new members.) In particular, it is not possible to join a members-only room without being on the member list, so an entity might need to request membership in order to join such a room.
If allowed, this functionality SHOULD be implemented by enabling a user to send a request for registration requirements to the room qualified by the 'jabber:iq:register' namespace as described in In-Band Registration (XEP-0077) [25]:

Listing 66: User Requests Registration Requirements

```
<iq from='hag66@shakespeare.lit/pda'
    id='jw81b36f'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='jabber:iq:register'/>
</iq>
```

If the room does not exist, the service MUST return an <item-not-found/> error.

Listing 67: Room Does Not Exist

```
<iq from='coven@chat.shakespeare.lit'
    id='jw81b36f'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the user requesting registration requirements is not allowed to register with the room (e.g., because that privilege has been restricted), the room MUST return a <not-allowed/> error to the user.

Listing 68: User Is Not Allowed to Register

```
<iq from='coven@chat.shakespeare.lit'
    id='jw81b36f'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
```

[25] XEP-0077: In-Band Registration <https://xmpp.org/extensions/xep-0077.html>.

```
  </error>
</iq>
```

If the user is already registered, as described in In-Band Registration (XEP-0077) [26] the room MUST reply with an IQ stanza of type "result", which MUST contain an empty <registered/> element and SHOULD contain at least a <username/> element that specifies the user's registered nickname in the room.

Listing 69: User Is Already Registered

```
<iq from='coven@chat.shakespeare.lit'
    id='jw81b36f'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='jabber:iq:register'>
    <registered/>
    <username>thirdwitch</username>
  </query>
</iq>
```

Otherwise, the room MUST then return a Data Form to the user (as described in Data Forms (XEP-0004) [27]). The information required to register might vary by implementation or deployment and is not fully specified in this document (e.g., the fields registered by this document for the 'http://jabber.org/protocol/muc#register' FORM_TYPE might be supplemented in the future via the mechanisms described in the Field Standardization section of this document). The following can be taken as a fairly typical example:

Listing 70: Service Returns Registration Form

```
<iq from='coven@chat.shakespeare.lit'
    id='jw81b36f'
    to='hag66@shakespeare.lit/pda'
    type='result'>
  <query xmlns='jabber:iq:register'>
    <instructions>
      To register on the web, visit http://shakespeare.lit/
    </instructions>
    <x xmlns='jabber:x:data' type='form'>
      <title>Dark Cave Registration</title>
      <instructions>
        Please provide the following information
        to register with this room.
      </instructions>
      <field
          type='hidden'
```

[26] XEP-0077: In-Band Registration <https://xmpp.org/extensions/xep-0077.html>.
[27] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

```
          var='FORM_TYPE'>
        <value>http://jabber.org/protocol/muc#register</value>
      </field>
      <field
          label='Given Name'
          type='text-single'
          var='muc#register_first'>
        <required/>
      </field>
      <field
          label='Family Name'
          type='text-single'
          var='muc#register_last'>
        <required/>
      </field>
      <field
          label='Desired Nickname'
          type='text-single'
          var='muc#register_roomnick'>
        <required/>
      </field>
      <field
          label='Your URL'
          type='text-single'
          var='muc#register_url'/>
      <field
          label='Email Address'
          type='text-single'
          var='muc#register_email'/>
      <field
          label='FAQ Entry'
          type='text-multi'
          var='muc#register_faqentry'/>
    </x>
  </query>
</iq>
```

The user SHOULD then submit the form:

Listing 71: User Submits Registration Form

```
<iq from='hag66@shakespeare.lit/pda'
    id='nv71va54'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='jabber:iq:register'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='FORM_TYPE'>
        <value>http://jabber.org/protocol/muc#register</value>
```

```
      </field>
      <field var='muc#register_first'>
        <value>Brunhilde</value>
      </field>
      <field var='muc#register_last'>
        <value>Entwhistle-Throckmorton</value>
      </field>
      <field var='muc#register_roomnick'>
        <value>thirdwitch</value>
      </field>
      <field var='muc#register_url'>
        <value>http://witchesonline/~hag66/</value>
      </field>
      <field var='muc#register_email'>
        <value>hag66@witchesonline</value>
      </field>
      <field var='muc#register_faqentry'>
        <value>Just another witch.</value>
      </field>
    </x>
  </query>
</iq>
```

If the desired room nickname is already reserved for that room, the room MUST return a <conflict/> error to the user:

Listing 72: Room Returns Conflict Error to User

```
<iq from='coven@chat.shakespeare.lit'
    id='nv71va54'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the room or service does not support registration, it MUST return a <service-unavailable/> error to the user:

Listing 73: Room Returns Service Unavailable Error to User

```
<iq from='coven@chat.shakespeare.lit'
    id='nv71va54'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error type='cancel'>
    <service-unavailable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
```

```
</iq>
```

If the user did not include a valid data form, the room MUST return a <bad-request/> error to the user:

Listing 74: Room Returns Service Bad Request Error to User

```
<iq from='coven@chat.shakespeare.lit'
    id='nv71va54'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Otherwise, the room MUST inform the user that the registration request was successfully received:

Listing 75: Room Informs User that Registration Request Has Been Processed

```
<iq from='coven@chat.shakespeare.lit'
    id='nv71va54'
    to='hag66@shakespeare.lit/pda'
    type='result'/>
```

After the user submits the form, the service MAY request that the submission be approved by a room admin/owner (see the Approving Registration Requests section of this document), MAY immediately add the user to the member list by changing the user's affiliation from "none" to "member", or MAY perform some service-specific checking (e.g., email verification).
If the service changes the user's affiliation and the user is in the room, it MUST send updated presence from this individual to all occupants, indicating the change in affiliation by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "member".

Listing 76: Service Sends Notice of Membership to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
  </x>
</presence>

[ ... ]
```

If the user's nickname is modified by the service as a result of registration and the user is in the room, the service SHOULD include status code "210" in the updated presence notification that it sends to all users.
If a user has registered with a room, the room MAY choose to restrict the user to use of the registered nickname only in that room. If it does so, it SHOULD modify the user's nickname to be the registered nickname (instead of returning a <not-acceptable/> error) if the user attempts to join the room with a roomnick other than the user's registered roomnick (this enables a room to "lock down" roomnicks for consistent identification of occupants).

7.11 Getting the Member List

If allowed in accordance with room configuration, an occupant MAY be allowed to retrieve the list of room members. For details, see the Modifying the Member List section of this document.

7.12 Discovering Reserved Room Nickname

A user MAY have a reserved room nickname, for example through explicit room registration, database integration, or nickname "lockdown". A user SHOULD discover his or her reserved nickname before attempting to enter the room. This is done by sending a Service Discovery information request to the room JID while specifying a well-known Service Discovery node of "x-roomuser-item".

Listing 77: User Requests Reserved Nickname

```
<iq from='hag66@shakespeare.lit/pda'
    id='getnick1'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='x-roomuser-item'/>
</iq>
```

It is OPTIONAL for a multi-user chat service to support the foregoing service discovery node. If the room or service does not support the foregoing service discovery node, it MUST return a <feature-not-implemented/> error to the user. If it does and the user has a registered nickname, it MUST return the nickname to the user as the value of the 'name' attribute of a Service Discovery <identity/> element (for which the category/type SHOULD be "conference/text"):

Listing 78: Room Returns Nickname

```
<iq from='coven@chat.shakespeare.lit'
    id='getnick1'
    to='hag66@shakespeare.lit/pda'
    type='result'>
```

```
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='x-roomuser-item'>
    <identity
        category='conference'
        name='thirdwitch'
        type='text'/>
  </query>
</iq>
```

If the user does not have a registered nickname, the room MUST return a service discovery <query/> element that is empty (in accordance with Service Discovery (XEP-0030) [28]).
Even if a user has registered one room nickname, the service SHOULD allow the user to specify a different nickname on entering the room (e.g., in order to join from different client resources), although the service MAY choose to "lock down" nicknames and therefore deny entry to the user, including a <not-acceptable/> error. The service MUST NOT return an error to the user if his or her client sends the foregoing request after having already joined the room, but instead SHOULD reply as previously described.
If another user attempts to join the room with a nickname reserved by the first user, the service MUST deny entry to the second user and return a <conflict/> error as previously described.

7.13 Requesting Voice

It is not possible for a visitor to speak (i.e., send a message to all occupants) in a moderated room. To request voice, a visitor SHOULD send a <message/> stanza containing a data form to the room itself, where the data form contains only a "muc#role" field with a value of "participant".

Listing 79: Occupant Requests Voice

```
<message from='hag66@shakespeare.lit/pda'
         id='yd53c486'
         to='coven@chat.shakespeare.lit'>
  <x xmlns='jabber:x:data' type='submit'>
    <field var='FORM_TYPE'>
      <value>http://jabber.org/protocol/muc#request</value>
    </field>
    <field var='muc#role'
           type='list-single'
           label='Requested role'>
      <value>participant</value>
    </field>
  </x>
</message>
```

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

The service then proceeds as described in the Approving Voice Requests section of this document.

7.14 Exiting a Room

In order to exit a multi-user chat room, an occupant sends a presence stanza of type ”unavailable” to the <room@service/nick> it is currently using in the room.

Listing 80: Occupant Exits a Room

```
<presence
    from='hag66@shakespeare.lit/pda'
    to='coven@chat.shakespeare.lit/thirdwitch'
    type='unavailable'/>
```

The service MUST then send a presence stanzas of type ”unavailable” from the departing user's occupant JID to the departing occupant's full JIDs, including a status code of ”110” to indicate that this notification is ”self-presence”:

Listing 81: Service Sends Self-Presence Related to Departure of Occupant

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='hag66@shakespeare.lit/pda'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='none'/>
    <status code='110'/>
  </x>
</presence>
```

Note: The presence stanza used to exit a room MUST possess a 'type' attribute whose value is ”unavailable”. For further discussion, see the Presence business rules.
The service MUST then send presence stanzas of type ”unavailable” from the departing user's occupant JID to the full JIDs of the remaining occupants:

Listing 82: Service Sends Presence Related to Departure of Occupant

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='hag66@shakespeare.lit/pda'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
```

```
          jid='hag66@shakespeare.lit/pda'
          role='none'/>
    <status code='110'/>
  </x>
</presence>
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='none'/>
  </x>
</presence>
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='wiccarocks@shakespeare.lit/laptop'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='none'/>
  </x>
</presence>
```

Presence stanzas of type "unavailable" reflected by the room MUST contain extended presence information about roles and affiliations; in particular, the 'role' attribute MUST be set to a value of "none" to denote that the individual is no longer an occupant.
The occupant MAY include normal <status/> information in the unavailable presence stanzas; this enables the occupant to provide a custom exit message if desired:

Listing 83: Custom Exit Message

```
<presence
    from='wiccarocks@shakespeare.lit/laptop'
    to='coven@chat.shakespeare.lit/oldhag'
    type='unavailable'>
  <status>gone where the goblins go</status>
</presence>
```

Normal presence stanza generation rules apply as defined in XMPP IM [29], so that if the user sends a general unavailable presence stanza, the user's server will broadcast that stanza to the client's <room@service/nick>; as a result, there is no need for the leaving client to send directed unavailable presence to its occupant JID. It is possible that a user might not be able to gracefully exit the room by sending unavailable presence. If the user goes offline

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

without sending unavailable presence, the user's server is responsible for sending unavailable presence on behalf of the user (in accordance with RFC 6121 [30]).
Note: See Ghost Users for suggestions regarding room occupants that appear to be present in the room but that are actually offline.
Note: If the room is not persistent and this occupant is the last to exit, the service is responsible for destroying the room.

8 Moderator Use Cases

A moderator has privileges to perform certain actions within the room (e.g., to change the roles of some occupants) but does not have rights to change persistent information about affiliations (which can be changed only by an admin or owner) or the room configuration. Exactly which actions can be performed by a moderator is subject to configuration. However, for the purposes of the MUC framework, moderators are stipulated to have privileges to perform the following actions:

1. discover an occupant's full JID in a semi-anonymous room (occurs automatically through presence)
2. modify the subject
3. kick a participant or visitor from the room
4. grant or revoke voice in a moderated room
5. modify the list of occupants who have voice in a moderated room

These features are implemented with a request/response exchange using <iq/> elements that contain child elements qualified by the 'http://jabber.org/protocol/muc#admin' namespace. The examples below illustrate the protocol interactions to implement the desired functionality. (Except where explicitly noted below, any of the following administrative requests MUST be denied if the <user@host> of the 'from' address of the request does not match the bare JID portion of one of the moderators; in this case, the service MUST return a <forbidden/> error.)

8.1 Modifying the Room Subject

A common feature of multi-user chat rooms is the ability to change the subject within the room.
By default, only users with a role of "moderator" SHOULD be allowed to change the subject in a room (although this is configurable, with the result that a mere participant or even visitor

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

might be allowed to change the subject, as controlled by the ”muc#roomconfig_changesubject” option).
The subject is changed by sending a message of type ”groupchat” to the <room@service>, where the <message/> MUST contain a <subject/> element that specifies the new subject but MUST NOT contain a <body/> element (or a <thread/> element). In accordance with the core definition of XMPP, other child elements are allowed (although the entity that receives them might ignore them).
Note: A message with a <subject/> and a <body/> or a <subject/> and a <thread/> is a legitimate message, but it SHALL NOT be interpreted as a subject change.

Listing 84: Moderator Changes Subject

```
<message
    from='wiccarocks@shakespeare.lit/laptop'
    id='lh2bs617'
    to='coven@chat.shakespeare.lit'
    type='groupchat'>
  <subject>Fire Burn and Cauldron Bubble!</subject>
</message>
```

The MUC service MUST reflect the message to all other occupants with a 'from' address equal to the room JID or to the occupant JID that corresponds to the sender of the subject change:

Listing 85: Service Informs All Occupants of Subject Change

```
<message
    from='coven@chat.shakespeare.lit/secondwitch'
    id='5BCE07C5-0729-4353-A6A3-ED9818C9B498'
    to='crone1@shakespeare.lit/desktop'
    type='groupchat'>
  <subject>Fire Burn and Cauldron Bubble!</subject>
</message>

[ ... ]
```

As explained under , when a new occupant joins the room the room SHOULD include the last subject change after the discussion history.
A MUC client that receives such a message MAY choose to display an in-room message, such as the following:

Listing 86: Client Displays Room Subject Change Message

```
* secondwitch has changed the subject to: Fire Burn and Cauldron
    Bubble!
```

If someone without appropriate privileges attempts to change the room subject, the service MUST return a message of type ”error” specifying a <forbidden/> error condition:

Listing 87: Service Returns Error Related to Unauthorized Subject Change

```
<message
    from='coven@chat.shakespeare.lit/thirdwitch'
    id='lh2bs617'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <subject>Fire Burn and Cauldron Bubble!</subject>
  <error by='coven@chat.shakespeare.lit' type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</message>
```

In order to remove the existing subject but not provide a new subject (i.e., set the subject to be empty), the client shall send an empty <subject/> element (i.e., either "<subject/>" or "<subject></subject>").

Listing 88: Moderator Sets Empty Subject

```
<message
    from='wiccarocks@shakespeare.lit/laptop'
    id='uj3bs61g'
    to='coven@chat.shakespeare.lit'
    type='groupchat'>
  <subject></subject>
</message>
```

8.2 Kicking an Occupant

A moderator has permissions to kick certain kinds of occupants from a room (which occupants are "kickable" depends on service provisioning, room configuration, and the moderator's affiliation -- see below). The kick is performed based on the occupant's room nickname and is completed by setting the role of a participant or visitor to a value of "none".

Listing 89: Moderator Kicks Occupant

```
<iq from='fluellen@shakespeare.lit/pda'
    id='kick1'
    to='harfleur@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='pistol' role='none'>
      <reason>Avaunt, you cullion!</reason>
    </item>
  </query>
</iq>
```

The service MUST remove the kicked occupant by sending a presence stanza of type ”unavailable” to each kicked occupant, including status code 307 in the extended presence information, optionally along with the reason (if provided) and the roomnick or bare JID of the user who initiated the kick.

Listing 90: Service Removes Kicked Occupant

```
<presence
    from='harfleur@chat.shakespeare.lit/pistol'
    to='pistol@shakespeare.lit/harfleur'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none'>
      <actor nick='Fluellen'/>
      <reason>Avaunt, you cullion!</reason>
    </item>
    <status code='110'/>
    <status code='307'/>
  </x>
</presence>
```

The inclusion of the status code assists clients in presenting their own notification messages (e.g., information appropriate to the user's locality). The optional inclusion of the reason and actor enable the kicked user to understand why he or she was kicked, and by whom if the kicked occupant would like to discuss the matter. [31]
After removing the kicked occupant(s), the service MUST then inform the moderator of success:

Listing 91: Service Informs Moderator of Success

```
<iq from='harfleur@chat.shakespeare.lit'
    id='kick1'
    to='fluellen@shakespeare.lit/pda'
    type='result'/>
```

After informing the moderator, the service MUST then inform all of the remaining occupants that the kicked occupant is no longer in the room by sending presence stanzas of type ”unavailable” from the individual's roomnick (<room@service/nick>) to all the remaining occupants (just as it does when occupants exit the room of their own volition), including the status code and optionally the reason and actor.

Listing 92: Service Informs Remaining Occupants

```
<presence
    from='harfleur@chat.shakespeare.lit/pistol'
```

[31] Some commentors have complained that this opens room owners and administrators up to potential abuse; unfortunately, with great power comes great responsibility.

```
    to='gower@shakespeare.lit/cell'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none'/>
    <status code='307'/>
  </x>
</presence>

[ ... ]
```

A user cannot be kicked by a moderator with a lower affiliation. Therefore, if a moderator who is a member attempts to kick an admin, or a moderator who is a member or admin attempts to kick an owner, the service MUST deny the request and return a <not-allowed/> error to the sender:

Listing 93: Service Returns Error on Attempt to Kick User With Higher Affiliation

```
<iq from='coven@chat.shakespeare.lit'
    id='kicktest'
    to='wiccarocks@shakespeare.lit/laptop'
    type='error'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If a moderator attempts to kick himself, the service MAY deny the request and return a <conflict/> error to the sender. (Although the act of kicking oneself may seem odd, it is common in IRC as a way of apologizing for one's actions in the room.)

8.3 Granting Voice to a Visitor

In a moderated room, a moderator might want to manage who does and does not have "voice" in the room (i.e., the ability to send messages to all occupants). Voice is granted based on the visitor's room nickname, which the service will convert into the visitor's full JID internally. The moderator grants voice to a visitor by changing the visitor's role to "participant".

Listing 94: Moderator Grants Voice to a Visitor

```
<iq from='crone1@shakespeare.lit/desktop'
    id='voice1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
          role='participant'/>
  </query>
```

```
</iq>
```

The <reason/> element is OPTIONAL:

Listing 95: Moderator Grants Voice to a Visitor (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='voice1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
          role='participant'>
      <reason>A worthy witch indeed!</reason>
    </item>
  </query>
</iq>
```

The service MUST then inform the moderator of success:

Listing 96: Service Informs Moderator of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='voice1'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST then send updated presence from this individual's <room@service/nick> to all occupants, indicating the addition of voice privileges by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "participant".

Listing 97: Service Sends Notice of Voice to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          nick='thirdwitch'
          role='participant'>
      <reason>A worthy witch indeed!</reason>
    </item>
  </x>
</presence>

[ ... ]
```

8.4 Revoking Voice from a Participant

In a moderated room, a moderator might want to revoke a participant's privileges to speak. The moderator can revoke voice from a participant by changing the participant's role to "visitor":

Listing 98: Moderator Revokes Voice from a Participant

```
<iq from='crone1@shakespeare.lit/desktop'
    id='voice2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
          role='visitor'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL:

Listing 99: Moderator Revokes Voice from a Visitor (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='voice2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
          role='visitor'>
      <reason>Not so worthy after all!</reason>
    </item>
  </query>
</iq>
```

The service MUST then inform the moderator of success:

Listing 100: Service Informs Moderator of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='voice2'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST then send updated presence from this individual to all occupants, indicating the removal of voice privileges by sending a presence element that contains an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "visitor".

Listing 101: Service Notes Loss of Voice

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='visitor'/>
  </x>
</presence>

[ ... ]
```

A moderator MUST NOT be able to revoke voice from a user whose affiliation is at or above the moderator's level. In addition, a service MUST NOT allow the voice privileges of an admin or owner to be removed by anyone. If a moderator attempts to revoke voice privileges from such a user, the service MUST deny the request and return a <not-allowed/> error to the sender along with the offending item(s):

Listing 102: Service Returns Error on Attempt to Revoke Voice from an Admin, Owner, or User with a Higher Affiliation

```
<iq from='coven@chat.shakespeare.lit'
    id='voicetest'
    to='crone1@shakespeare.lit/desktop'
    type='error'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.5 Modifying the Voice List

A moderator in a moderated room might want to modify the voice list. To do so, the moderator first requests the voice list by querying the room for all occupants with a role of 'participant'.

Listing 103: Moderator Requests Voice List

```
<iq from='bard@shakespeare.lit/globe'
    id='voice3'
    to='goodfolk@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item role='participant'/>
  </query>
</iq>
```

The service MUST then return the voice list to the moderator; each item MUST include the 'nick' and 'role' attributes and SHOULD include the 'affiliation' and 'jid' attributes:

Listing 104: Service Sends Voice List to Moderator

```
<iq from='goodfolk@chat.shakespeare.lit'
    id='voice3'
    to='bard@shakespeare.lit/globe'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='none'
          jid='polonius@hamlet/castle'
          nick='Polo'
          role='participant'/>
    <item affiliation='none'
          jid='horatio@hamlet/castle'
          nick='horotoro'
          role='participant'/>
    <item affiliation='member'
          jid='hecate@shakespeare.lit/broom'
          nick='Hecate'
          role='participant'/>
  </query>
</iq>
```

The moderator can then modify the voice list if desired. In order to do so, the moderator MUST send the changed items (i.e., only the "delta") back to the service; each item MUST include the 'nick' attribute and 'role' attribute (normally set to a value of "participant" or "visitor") but SHOULD NOT include the 'jid' attribute and MUST NOT include the 'affiliation' attribute (which is used to manage affiliations such as owner rather than the participant role):

Listing 105: Moderator Sends Modified Voice List to Service

```
<iq from='bard@shakespeare.lit/globe'
    id='voice4'
    to='goodfolk@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='Hecate'
          role='visitor'/>
    <item nick='rosencrantz'
          role='participant'>
      <reason>A worthy fellow.</reason>
    </item>
    <item nick='guildenstern'
          role='participant'>
      <reason>A worthy fellow.</reason>
    </item>
```

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

The service MUST then inform the moderator of success:

Listing 106: Service Informs Moderator of Success

```
<iq from='goodfolk@chat.shakespeare.lit'
    id='voice1'
    to='bard@shakespeare.lit/globe'
    type='result'/>
```

The service MUST then send updated presence for any affected individuals to all occupants, indicating the change in voice privileges by sending the appropriate extended presence stanzas as described in the foregoing use cases.
As noted, voice privileges cannot be revoked from a room owner or room admin, nor from any user with a higher affiliation than the moderator making the request. If a room admin attempts to revoke voice privileges from such a user by modifying the voice list, the service MUST deny the request and return a <not-allowed/> error to the sender:

Listing 107: Service Returns Error on Attempt to Revoke Voice from an Admin, Owner, or User with a Higher Affiliation

```
<iq from='goodfolk@chat.shakespeare.lit'
    id='voicetest'
    to='bard@shakespeare.lit/globe'
    type='error'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.6 Approving Voice Requests

As noted in the Requesting Voice section of this document, an occupant requests voice by sending a voice request data form to the service. The service then SHOULD use that voice request data form as the basis for a voice approval data form that it generates and sends to the room moderator(s). The voice approval data form is contained in a <message/> stanza, as shown below.

Listing 108: Voice Request Approval Form

```
<message from='coven@chat.shakespeare.lit'
         id='approve'
         to='crone1@shakespeare.lit/pda'>
  <x xmlns='jabber:x:data' type='form'>
    <title>Voice request</title>
```

```
    <instructions>
      To approve this request for voice, select
      the &quot;Grant voice to this person?&quot;
      checkbox and click OK. To skip this request,
      click the cancel button.
    </instructions>
    <field var='FORM_TYPE' type='hidden'>
        <value>http://jabber.org/protocol/muc#request</value>
    </field>
    <field var='muc#role'
           type='list-single'
           label='Requested role'>
      <value>participant</value>
    </field>
    <field var='muc#jid'
           type='jid-single'
           label='User ID'>
      <value>hag66@shakespeare.lit/pda</value>
    </field>
    <field var='muc#roomnick'
           type='text-single'
           label='Room Nickname'>
      <value>thirdwitch</value>
    </field>
    <field var='muc#request_allow'
           type='boolean'
           label='Grant voice to this person?'>
      <value>false</value>
    </field>
  </x>
</message>
```

In order to approve the request, a moderator shall submit the form:

Listing 109: Voice Request Approval Submission

```
<message from='crone1@shakespeare.lit/pda'
         id='approve'
         to='coven@chat.shakespeare.lit'>
  <x xmlns='jabber:x:data' type='submit'>
    <field var='FORM_TYPE' type='hidden'>
        <value>http://jabber.org/protocol/muc#request</value>
    </field>
    <field var='muc#role'>
      <value>participant</value>
    </field>
    <field var='muc#jid'>
      <value>hag66@shakespeare.lit/pda</value>
    </field>
```

```
    <field var='muc#roomnick'>
      <value>thirdwitch</value>
    </field>
    <field var='muc#request_allow'>
      <value>true</value>
    </field>
  </x>
</message>
```

If a moderator approves the voice request, the service shall grant voice to the occupant and send a presence update as described in the Granting Voice to a Visitor section of this document.

9 Admin Use Cases

A room administrator has privileges to modify persistent information about user affiliations (e.g., by banning users) and to grant and revoke moderator status, but does not have rights to change the room configuration, which is the sole province of the room owner(s). Exactly which actions can be performed by a room admin is subject to configuration. However, for the purposes of the MUC framework, room admins are stipulated to at a minimum have privileges to perform the following actions:

1. ban a user from the room
2. modify the list of users who are banned from the room
3. grant or revoke membership
4. modify the member list
5. grant or revoke moderator status
6. modify the list of moderators

These features are implemented with a request/response exchange using <iq/> elements that contain child elements qualified by the 'http://jabber.org/protocol/muc#admin' namespace. The examples below illustrate the protocol interactions that implement the desired functionality. (Except where explicitly noted below, any of the following administrative requests MUST be denied if the <user@host> of the 'from' address of the request does not match the bare JID of one of the room admins; in this case, the service MUST return a <forbidden/> error.)

9.1 Banning a User

An admin or owner can ban one or more users from a room. The ban MUST be performed based on the occupant's bare JID. In order to ban a user, an admin MUST change the user's

affiliation to ”outcast”.

Listing 110: Admin Bans User

```
<iq from='kinghenryv@shakespeare.lit/throne'
    id='ban1'
    to='southampton@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='outcast'
          jid='earlofcambridge@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 111: Admin Bans User (With a Reason)

```
<iq from='kinghenryv@shakespeare.lit/throne'
    id='ban1'
    to='southampton@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='outcast'
          jid='earlofcambridge@shakespeare.lit'>
      <reason>Treason</reason>
    </item>
  </query>
</iq>
```

The service MUST add that bare JID to the ban list, MUST remove the outcast’s nickname from the list of registered nicknames, and MUST inform the admin or owner of success:

Listing 112: Service Informs Admin or Owner of Success

```
<iq from='southampton@chat.shakespeare.lit'
    id='ban1'
    to='kinghenryv@shakespeare.lit/throne'
    type='result'/>
```

The service MUST also remove any banned users who are in the room by sending a presence stanza of type ”unavailable” to each banned occupant, including status code 301 in the extended presence information, optionally along with the reason (if provided) and the roomnick or bare JID of the user who initiated the ban.

Listing 113: Service Removes Banned User

```
<presence
```

```
    from='southampton@chat.shakespeare.lit/cambridge'
    to='earlofcambridge@shakespeare.lit/stabber'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='outcast' role='none'>
      <actor nick='The␣ '/>
      <reason>Treason</reason>
    </item>
    <status code='301'/>
  </x>
</presence>
```

The inclusion of the status code assists clients in presenting their own notification messages (e.g., information appropriate to the user's locality). The optional inclusion of the reason and actor enable the banned user to understand why he or she was banned, and by whom if the banned user would like to discuss the matter.

The service MUST then inform all of the remaining occupants that the banned user is no longer in the room by sending presence stanzas of type "unavailable" from the banned user to all remaining occupants (just as it does when occupants exit the room of their own volition), including the status code and optionally the reason and actor:

Listing 114: Service Informs Remaining Occupants

```
<presence
    type='unavailable'
    from='southampton@chat.shakespeare.lit/cambridge'
    to='exeter@shakespeare.lit/pda'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='outcast'
          jid='earlofcambridge@shakespeare.lit/stabber'
          role='none'/>
    <status code='301'/>
  </x>
</presence>

[ ... ]
```

As with Kicking an Occupant, a user cannot be banned by an admin with a lower affiliation. Therefore, if an admin attempts to ban an owner, the service MUST deny the request and return a <not-allowed/> error to the sender:

Listing 115: Service Returns Error on Attempt to Ban User With Higher Affiliation

```
<iq from='kinghenryv@shakespeare.lit/throne'
    id='ban1'
    to='southampton@chat.shakespeare.lit'
    type='set'>
  <error type='cancel'>
```

```
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If an admin or owner attempts to ban himself, the service MUST deny the request and return a <conflict/> error to the sender. (Note: This is different from the recommended service behavior on kicking oneself.)

9.2 Modifying the Ban List

A room admin might want to modify the ban list. (Note: The ban list is always based on a user's bare JID.) To modify the list of banned JIDs, the admin first requests the ban list by querying the room for all users with an affiliation of 'outcast'.

Listing 116: Admin Requests Ban List

```
<iq from='kinghenryv@shakespeare.lit/throne'
    id='ban2'
    to='southampton@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='outcast'/>
  </query>
</iq>
```

The service MUST then return the list of banned users to the admin; each item MUST include the 'affiliation' and 'jid' attributes but SHOULD NOT include the 'nick' and 'role' attributes:

Listing 117: Service Sends Ban List to Admin

```
<iq from='southampton@chat.shakespeare.lit'
    id='ban2'
    to='kinghenryv@shakespeare.lit/throne'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='outcast'
          jid='earlofcambridge@shakespeare.lit'>
      <reason>Treason</reason>
    </item>
  </query>
</iq>
```

The admin can then modify the ban list if desired. In order to do so, the admin MUST send the changed items (i.e., only the "delta") back to the service; each item MUST include the 'affiliation' attribute (normally set to a value of "outcast" to ban or "none" to remove ban) and 'jid' attribute but SHOULD NOT include the 'nick' attribute and MUST NOT include the 'role' attribute (which is used to manage roles such as participant rather than affiliations such

as outcast); in addition, the reason and actor elements are OPTIONAL:

Listing 118: Admin Sends Modified Ban List to Service

```
<iq from='kinghenryv@shakespeare.lit/throne'
    id='ban3'
    to='southampton@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='outcast'
          jid='lordscroop@shakespeare.lit'>
      <reason>Treason</reason>
    </item>
    <item affiliation='outcast'
          jid='sirthomasgrey@shakespeare.lit'>
      <reason>Treason</reason>
    </item>
  </query>
</iq>
```

After updating the ban list, the service MUST inform the admin of success:

Listing 119: Service Informs Admin of Success

```
<iq from='southampton@chat.shakespeare.lit'
    id='ban3'
    to='kinghenryv@shakespeare.lit/throne'
    type='result'/>
```

The service MUST then remove the affected occupants (if they are in the room) and send updated presence (including the appropriate status code) from them to all the remaining occupants as described in the "Banning a User" use case. (The service MUST also remove each banned user's reserved nickname from the list of reserved roomnicks, if appropriate.)
When an entity is banned from a room, an implementation SHOULD match JIDs in the following order:

1. <user@domain>
2. <domain> (the domain itself matches, as does any user@domain)

Some administrators might wish to ban all users associated with a specific domain from all rooms hosted by a MUC service. Such functionality is a service-level feature and is therefore out of scope for this document; see Service Administration (XEP-0133) [32].
As specified in Banning a User, users cannot be banned under certain conditions. For example: admins and owners cannot ban themselves, and a user cannot be banned by an admin with a

[32] XEP-0133: Service Administration <https://xmpp.org/extensions/xep-0133.html>.

lower affiliation. When a request to modify the ban list includes one or more modifications that is prohibited by the definitions in Banning a User, then the service SHOULD NOT apply any of the requested changes and MUST deny the request using an error which SHOULD be either <conflict/> or <not-allowed/>.

9.3 Granting Membership

An admin can grant membership to a user; this is done by changing the affiliation for the user's bare JID to "member" (if a nick is provided, that nick becomes the user's default nick in the room if that functionality is supported by the implementation):

Listing 120: Admin Grants Membership

```
<iq from='crone1@shakespeare.lit/desktop'
    id='member1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit'
          nick='thirdwitch'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 121: Admin Grants Membership (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='member1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit'
          nick='thirdwitch'>
      <reason>A worthy witch indeed!</reason>
    </item>
  </query>
</iq>
```

The service MUST add the user to the member list and then inform the admin of success:

Listing 122: Service Informs Admin of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='member1'
```

```
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

If the user is in the room, the service MUST then send updated presence from this individual to all occupants, indicating the granting of membership by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "member".

Listing 123: Service Sends Notice of Membership to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='participant'
          nick='thirdwitch'/>
  </x>
</presence>

[ ... ]
```

9.4 Revoking Membership

An admin might want to revoke a user's membership; this is done by changing the user's affiliation to "none":

Listing 124: Admin Revokes Membership

```
<iq from='crone1@shakespeare.lit/desktop'
    id='member2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='none'
          jid='hag66@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 125: Admin Revokes Membership (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='member2'
    to='coven@chat.shakespeare.lit'
```

```
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='none'
          jid='hag66@shakespeare.lit'>
      <reason>Not so worthy after all!</reason>
    </item>
  </query>
</iq>
```

The service MUST remove the user from the member list and then inform the moderator of success:

Listing 126: Service Informs Moderator of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='member2'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST then send updated presence from this individual to all occupants, indicating the loss of membership by sending a presence element that contains an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "none".

Listing 127: Service Notes Loss of Membership

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
  </x>
</presence>

[ ... ]
```

If the room is members-only, the service MUST remove the user from the room, including a status code of 321 to indicate that the user was removed because of an affiliation change, and inform all remaining occupants. The stanza MAY include an <actor/> element.

Listing 128: Service Removes Non-Member

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
    type='unavailable'>
```

```
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none'>
      <actor nick='TheBard'/>
    </item>
    <status code='321'/>
  </x>
</presence>

[ ... ]
```

As there is no point in time where a non-member user must be in a members-only room, the service SHOULD NOT send both a de-affiliation presence (without a 'type' attribute) followed by room-removal presence (of type 'unavailable'). Instead, it SHOULD only send the latter of the two.

9.5 Modifying the Member List

In the context of a members-only room, the member list is essentially a list of people who are allowed to enter the room. Anyone who is not a member is effectively banned from entering the room, even if their affiliation is not "outcast".
In the context of an open room, the member list is simply a list of users (bare JID and reserved nick) who are registered with the room. Such users can appear in a room roster, have their room nickname reserved, be returned in search results or FAQ queries, and the like.
It is RECOMMENDED that only room admins have the privilege to modify the member list in members-only rooms. To do so, the admin first requests the member list by querying the room for all users with an affiliation of "member":

Listing 129: Admin Requests Member List

```
<iq from='crone1@shakespeare.lit/desktop'
    id='member3'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='member'/>
  </query>
</iq>
```

Note: If the room is non-anonymous, a service SHOULD also return the member list to any occupant in a members-only room; i.e., it SHOULD NOT generate a <forbidden/> error when a member in such a room requests the member list. This functionality can assist clients in showing all the existing members even if some of them are not in the room, e.g. to help a member determine if another user should be invited. If the room is non-anonymous, a service SHOULD also allow any member to retrieve the member list even if not yet an occupant.
The service MUST then return the full member list to the admin qualified by the

'http://jabber.org/protocol/muc#admin' namespace; each item MUST include the 'affiliation' and 'jid' attributes and MAY include the 'nick' and 'role' attributes for each member that is currently an occupant.

Listing 130: Service Sends Member List to Admin

```
<iq from='coven@chat.shakespeare.lit'
    id='member3'
    to='crone1@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit'
          nick='thirdwitch'
          role='participant'/>
  </query>
</iq>
```

The admin can then modify the member list if desired. In order to do so, the admin MUST send the changed items (i.e., only the "delta") to the service; each item MUST include the 'affiliation' attribute (normally set to a value of "member" or "none") and 'jid' attribute but SHOULD NOT include the 'nick' attribute (unless modifying the user's reserved nickname) and MUST NOT include the 'role' attribute (which is used to manage roles such as participant rather than affiliations such as member):

Listing 131: Admin Sends Modified Member List to Service

```
<iq from='crone1@shakespeare.lit/desktop'
    id='member4'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='none'
          jid='hag66@shakespeare.lit'/>
    <item affiliation='member'
          jid='hecate@shakespeare.lit'/>
  </query>
</iq>
```

The service MUST modify the member list and then inform the moderator of success:

Listing 132: Service Informs Moderator of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='member4'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST change the affiliation of any affected user. If the user has been removed from the member list, the service MUST change the user's affiliation from "member" to "none". If the user has been added to the member list, the service MUST change the user's affiliation to "member".
If a removed member is currently in a members-only room, the service SHOULD kick the occupant by changing the removed member's role to "none" and send appropriate presence to the removed member as previously described. The service MUST subsequently refuse entry to the user.
For all room types, the service MUST send updated presence from this individual to all occupants, indicating the change in affiliation by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "none".

Listing 133: Service Sends Notice of Loss of Membership to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
  </x>
</presence>

[ ... ]
```

In addition, the service SHOULD send an invitation to any user who has been added to the member list of a members-only room if the user is not currently affiliated with the room (note that the following example includes a password but not a reason -- both child elements are OPTIONAL):

Listing 134: Room Sends Invitation to New Member

```
<message
    from='coven@chat.shakespeare.lit'
    id='CA409450-5AAE-41C1-AAAD-5375CA738885'
    to='hecate@shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite from='bard@shakespeare.lit'/>
    <password>cauldronburn</password>
  </x>
</message>
```

Although only admins and owners SHOULD be allowed to modify the member list, an implementation MAY provide a configuration option that opens invitation privileges to any member of a members-only room. In such a situation, any invitation sent SHOULD automati-

cally trigger the addition of the invitee to the member list. However, if invitation privileges are restricted to admins and a mere member attempts to a send an invitation, the service MUST deny the invitation request and return a <forbidden/> error to the sender:

Listing 135: Service Returns Error on Attempt by Mere Member to Invite Others to a Members-Only Room

```
<message
    from='coven@chat.shakespeare.lit'
    id='C6E14DF6-00B7-4729-BC1C-94E59C07548E'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite to='hecate@shakespeare.lit'>
      <reason>
        Hey Hecate, this is the place for all good witches!
      </reason>
    </invite>
  </x>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</message>
```

Invitations sent through an open room MUST NOT trigger the addition of the invitee to the member list.
If a user is added to the member list of an open room and the user is in the room, the service MUST send updated presence from this individual to all occupants, indicating the change in affiliation by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "member".

Listing 136: Service Sends Notice of Membership to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/hecate'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hecate@shakespeare.lit/broom'
          role='participant'/>
  </x>
</presence>

[ ... ]
```

9.6 Granting Moderator Status

An admin might want to grant moderator status to a participant or visitor; this is done by changing the user's role to "moderator":

Listing 137: Admin Grants Moderator Status

```
<iq from='crone1@shakespeare.lit/desktop'
    id='mod1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
          role='moderator'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 138: Admin Grants Moderator Status (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='mod1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
          role='moderator'>
      <reason>A worthy witch indeed!</reason>
    </item>
  </query>
</iq>
```

The service MUST add the user to the moderator list and then inform the admin of success:

Listing 139: Service Informs Admin of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='mod1'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST then send updated presence from this individual to all occupants, indicating the addition of moderator status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "moderator".

Listing 140: Service Sends Notice of Moderator Status to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='moderator'/>
  </x>
</presence>

[ ... ]
```

9.7 Revoking Moderator Status

An admin might want to revoke a user's moderator status. An admin is allowed to revoke moderator status only from a user whose affiliation is "member" or "none" (i.e., not from an owner or admin). The status is revoked by changing the user's role to "participant":

Listing 141: Admin Revokes Moderator Status

```
<iq from='crone1@shakespeare.lit/desktop'
    id='mod2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
          role='participant'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 142: Admin Revokes Moderator Status (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='mod2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
          role='participant'>
      <reason>Not so worthy after all!</reason>
    </item>
  </query>
</iq>
```

The service MUST remove the user from the moderator list and then inform the admin of success:

Listing 143: Service Informs Admin of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='mod2'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST then send updated presence from this individual to all occupants, indicating the removal of moderator status by sending a presence element that contains an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "participant".

Listing 144: Service Notes Loss of Moderator Status

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          role='participant'/>
  </x>
</presence>

[ ... ]
```

As noted, an admin MUST NOT be allowed to revoke moderator status from a user whose affiliation is "owner" or "admin". If an admin attempts to revoke moderator status from such a user, the service MUST deny the request and return a <not-allowed/> error to the sender:

Listing 145: Service Returns Error on Attempt to Revoke Moderator Status from an Admin or Owner

```
<iq from='coven@chat.shakespeare.lit'
    id='modtest'
    to='crone1@shakespeare.lit/desktop'
    type='error'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

9.8 Modifying the Moderator List

An admin might want to modify the moderator list. To do so, the admin first requests the moderator list by querying the room for all users with a role of 'moderator'.

Listing 146: Admin Requests Moderator List

```
<iq from='crone1@shakespeare.lit/desktop'
    id='mod3'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item role='moderator'/>
  </query>
</iq>
```

The service MUST then return the moderator list to the admin; each item MUST include the 'nick' and 'role' attributes, and MAY include the 'jid' and 'affiliation' attributes:

Listing 147: Service Sends Moderator List to Admin

```
<iq from='coven@chat.shakespeare.lit'
    id='mod3'
    to='crone1@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='member'
          jid='hag66@shakespeare.lit/pda'
          nick='thirdwitch'
          role='moderator'/>
  </query>
</iq>
```

The admin can then modify the moderator list if desired. In order to do so, the admin MUST send the changed items (i.e., only the "delta") back to the service; each item MUST include the 'nick' attribute and 'role' attribute (set to a value of "moderator" to grant moderator status or "participant" to revoke moderator status), but SHOULD NOT include the 'jid' attribute and MUST NOT include the 'affiliation' attribute (which is used to manage affiliations such as admin rather than the moderator role):

Listing 148: Admin Sends Modified Moderator List to Service

```
<iq from='crone1@shakespeare.lit/desktop'
    id='mod4'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item nick='thirdwitch'
```

```
          role='participant'/>
    <item nick='Hecate'
          role='moderator'/>
  </query>
</iq>
```

The service MUST modify the moderator list and then inform the admin of success:

Listing 149: Service Informs Admin of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='mod4'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST then send updated presence for any affected individuals to all occupants, indicating the change in moderator status by sending the appropriate extended presence stanzas as described in the foregoing use cases.
As noted, moderator status cannot be revoked from a room owner or room admin. If a room admin attempts to revoke moderator status from such a user by modifying the moderator list, the service MUST deny the request and return a <not-allowed/> error to the sender:

Listing 150: Service Returns Error on Attempt to Revoke Moderator Status from an Admin or Owner

```
<iq from='coven@chat.shakespeare.lit'
    id='modtest'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

9.9 Approving Registration Requests

If a service does not automatically accept requests to register with a room, it MAY provide a way for room admins to approve or deny registration requests over XMPP (alternatively, it could provide a web interface or some other admin tool). The simplest way to do so is for the service to send a <message/> stanza to the room admin(s) when the registration request is received, where the <message/> stanza contains a Data Form asking for approval or denial of the request. The following Data Form is RECOMMENDED but implementations might use a different form entirely, or supplement the following form with additional fields.

Listing 151: Registration Request Approval Form

```
<message from='coven@chat.shakespeare.lit'
         id='407665A9-E54E-4AD5-905F-9FD8864489B3'
         to='crone1@shakespeare.lit/pda'>
  <x xmlns='jabber:x:data' type='form'>
    <title>Registration request</title>
    <instructions>
      To approve this registration request, select the
      &quot;Allow this person to register with the room?&quot;
      checkbox and click OK. To skip this request, click the
      cancel button.
    </instructions>
    <field var='FORM_TYPE' type='hidden'>
        <value>http://jabber.org/protocol/muc#register</value>
    </field>
    <field var='muc#register_first'
           type='text-single'
           label='Given Name'>
      <value>Brunhilde</value>
    </field>
    <field var='muc#register_last'
           type="text-single"
           label="Family Name">
      <value>Entwhistle-Throckmorton</value>
    </field>
    <field var='muc#register_roomnick'
           type="text-single"
           label="Desired Nickname">
      <value>thirdwitch</value>
    </field>
    <field var='muc#register_url'
           type="text-single"
           label="User URL">
      <value>http://witchesonline/~hag66/</value>
    </field>
    <field var='muc#register_email'
           type="text-single"
           label="Email Address">
      <value>hag66@witchesonline</value>
    </field>
    <field var='muc#register_faqentry'
           type="text-multi"
           label="FAQ Entry">
      <value>Just another witch.</value>
    </field>
    <field var='muc#register_allow'
           type='boolean'
           label='Allow this person to register with the room?'>
      <value>0</value>
    </field>
```

```
  </x>
</message>
```

If the admin approves the registration request, the service shall register the user with the room.
More advanced registration approval mechanisms (e.g., retrieving a list of registration requests using Ad-Hoc Commands (XEP-0050) [33] as is done in Publish-Subscribe (XEP-0060) [34]) are out of scope for this document.

10 Owner Use Cases

Every room MUST have at least one owner, and that owner (or a successor) is a long-lived attribute of the room for as long as the room exists (e.g., the owner does not lose ownership on exiting a persistent room). This document assumes that the (initial) room owner is the individual who creates the room and that only a room owner has the right to change defining room configuration settings such as the room type. Room owners can specify not only the room types (password-protected, members-only, etc.) but also certain attributes of the room as listed in the Requirements section of this document. In addition, an owner can also specify the JIDs of other owners, if supported by the implementation.
In order to provide the necessary flexibility for a wide range of configuration options, Data Forms (Data Forms (XEP-0004) [35]) are used for room configuration. The service shall require configuration via Data Forms before creating and unlocking the room.
Note: The configuration options shown below address all of the features and room types listed in the requirements section of this document; however, the exact configuration options and form layout shall be determined by the implementation or specific deployment. Also, these are examples only and are not intended to define the only allowed or required configuration options for rooms. A given implementation or deployment MAY choose to provide additional configuration options (clearance levels, profanity filters, supported languages, message logging, etc.), which is why the use of the 'jabber:x:data' protocol is valuable here.

10.1 Creating a Room

10.1.1 General Considerations

The privilege to create rooms MAY be restricted to certain users or MAY be reserved to an administrator of the service. If access is restricted and a user attempts to create a room, the service MUST return a <not-allowed/> error:

[33] XEP-0050: Ad-Hoc Commands <https://xmpp.org/extensions/xep-0050.html>.
[34] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[35] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

Listing 152: Service Informs User of Inability to Create a Room

```
<presence
    from='coven@chat.shakespeare.lit/thirdwitch'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <x xmlns='http://jabber.org/protocol/muc'/>
  <error by='coven@chat.shakespeare.lit' type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

If access is not restricted, the service MUST allow the user to create a room as described below.
From the perspective of room creation, there are in essence two kinds of rooms:

- "Instant rooms" -- these are available for immediate access and are automatically created based on some default configuration.

- "Reserved rooms" -- these are manually configured by the room creator before anyone is allowed to enter.

The workflow for creating and configuring such rooms is as follows:

1. The user sends presence to <room@service/nick> and signal his or her support for the Multi-User Chat protocol by including extended presence information in an empty <x/> child element qualified by the 'http://jabber.org/protocol/muc' namespace (note the lack of an '#owner' or '#user' fragment).

2. If this user is allowed to create a room and the room does not yet exist, the service MUST create the room according to some default configuration, assign the requesting user as the initial room owner, and add the owner to the room but not allow anyone else to enter the room (effectively "locking" the room). The initial presence stanza received by the owner from the room MUST include extended presence information indicating the user's status as an owner and acknowledging that the room has been created (via status code 201) and is awaiting configuration.

3. If the initial room owner would like to create and configure a reserved room, the room owner MUST then request a configuration form by sending an IQ stanza of type "get" to the room containing an empty <query/> element qualified by the 'http://jabber.org/protocol/muc#owner' namespace, then complete Steps 4 and 5. If the room owner would prefer to create an instant room, the room owner MUST send

a query element qualified by the 'http://jabber.org/protocol/muc#owner' namespace and containing an empty <x/> element of type "submit" qualified by the 'jabber:x:data' namespace, then skip to Step 6.

4. If the room owner requested a configuration form, the service MUST send an IQ result to the room owner containing a configuration form qualified by the 'jabber:x:data' namespace. If there are no configuration options available, the room MUST return an empty query element to the room owner.

5. The initial room owner SHOULD provide a starting configuration for the room (or accept the default configuration) by sending an IQ set containing the completed configuration form. Alternatively, the room owner MAY cancel the configuration process. (An implementation MAY set a timeout for initial configuration, such that if the room owner does not configure the room within the timeout period, the room owner is assumed to have accepted the default configuration or to have cancelled the configuration process.)

6. Once the service receives the completed configuration form from the initial room owner (or receives a request for an instant room), the service MUST "unlock" the room (i.e., allow other users to enter the room) and send an IQ of type "result" to the room owner. If the service receives a cancellation, it MUST destroy the room.

The protocol for this workflow is shown in the examples below.
First, the user MUST send presence to the room, including an empty <x/> element qualified by the 'http://jabber.org/protocol/muc' namespace (this is the same stanza sent when seeking to enter a room).

Listing 153: User Creates a Room and Signals Support for Multi-User Chat

```
<presence
    from='crone1@shakespeare.lit/desktop'
    to='coven@chat.shakespeare.lit/firstwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>
```

If the room does not yet exist, the service SHOULD create the room (subject to local policies regarding room creation), assign the bare JID of the requesting user as the owner, add the owner to the room, and acknowledge successful creation of the room by sending a presence stanza of the following form:

Listing 154: Service Acknowledges Room Creation

```
<presence
```

```
    from='coven@chat.shakespeare.lit/firstwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='owner'
          role='moderator'/>
    <status code='110'/>
    <status code='201'/>
  </x>
</presence>
```

After receiving notification that the room has been created, the room owner needs to decide whether to accept the default room configuration (i.e., create an "instant room") or configure the room to use something other than the default room configuration (i.e., create a "reserved room"). The protocol flows for completing those two use cases are shown in the following sections.

10.1.2 Creating an Instant Room

If the initial room owner wants to accept the default room configuration (i.e., create an "instant room"), the room owner MUST decline an initial configuration form by sending an IQ set to the <room@service> itself containing a <query/> element qualified by the 'http://jabber.org/protocol/muc#owner' namespace, where the only child of the <query/> is an empty <x/> element that is qualified by the 'jabber:x:data' namespace and that possesses a 'type' attribute whose value is "submit":

Listing 155: Owner Requests Instant Room

```
<iq from='crone1@shakespeare.lit/desktop'
    id='create1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <x xmlns='jabber:x:data' type='submit'/>
  </query>
</iq>
```

The service MUST then unlock the room and allow other entities to join it.

10.1.3 Creating a Reserved Room

If the initial room owner wants to create and configure a reserved room, the room owner MUST request an initial configuration form by sending an IQ get to the <room@service> itself containing an empty <query/> element qualified by the 'http://jabber.org/protocol/muc#owner' namespace:

Listing 156: Owner Requests Configuration Form

```
<iq from='crone1@shakespeare.lit/desktop'
    id='create1'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/muc#owner'/>
</iq>
```

If the room does not already exist, the service MUST return an initial room configuration form to the user. (Note: The following example shows a representative sample of configuration options. A full list of x:data fields registered for use in room creation and configuration is maintained by the XMPP Registrar; see the XMPP Registrar Considerations section of this document.)

Listing 157: Service Sends Configuration Form

```
<iq from='coven@chat.shakespeare.lit'
    id='create1'
    to='crone1@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <x xmlns='jabber:x:data' type='form'>
      <title>Configuration for "coven" Room</title>
      <instructions>
          Your room coven@macbeth has been created!
          The default configuration is as follows:
            - No logging
            - No moderation
            - Up to 20 occupants
            - No password required
            - No invitation required
            - Room is not persistent
            - Only admins may change the subject
            - Presence broadcasted for all users
          To accept the default configuration, click OK. To
          select a different configuration, please complete
          this form.
      </instructions>
      <field
          type='hidden'
          var='FORM_TYPE'>
        <value>http://jabber.org/protocol/muc#roomconfig</value>
      </field>
      <field
          label='Natural-Language␣Room␣Name'
          type='text-single'
          var='muc#roomconfig_roomname'/>
      <field
          label='Short␣Description␣of␣Room'
```

```
        type='text-single'
        var='muc#roomconfig_roomdesc'/>
    <field
        label='Natural Language for Room Discussions'
        type='text-single'
        var='muc#roomconfig_lang'/>
    <field
        label='Enable Public Logging?'
        type='boolean'
        var='muc#roomconfig_enablelogging'>
      <value>0</value>
    </field>
    <field
        label='Allow Occupants to Change Subject?'
        type='boolean'
        var='muc#roomconfig_changesubject'>
      <value>0</value>
    </field>
    <field
        label='Allow Occupants to Invite Others?'
        type='boolean'
        var='muc#roomconfig_allowinvites'>
      <value>0</value>
    </field>
    <field
        label='Who Can Send Private Messages?'
        type='list-single'
        var='muc#roomconfig_allowpm'>
      <value>anyone</value>
      <option label='Anyone'>
        <value>anyone</value>
      </option>
      <option label='Anyone with Voice'>
        <value>participants</value>
        </option>
      <option label='Moderators Only'>
        <value>moderators</value>
      </option>
      <option label='Nobody'>
        <value>none</value>
      </option>
    </field>
    <field
        label='Maximum Number of Occupants'
        type='list-single'
        var='muc#roomconfig_maxusers'>
      <value>20</value>
      <option label='10'><value>10</value></option>
      <option label='20'><value>20</value></option>
```

```
      <option label='30'><value>30</value></option>
      <option label='50'><value>50</value></option>
      <option label='100'><value>100</value></option>
      <option label='None'><value>none</value></option>
    </field>
    <field
        label='Roles␣for␣which␣Presence␣is␣Broadcasted'
        type='list-multi'
        var='muc#roomconfig_presencebroadcast'>
      <value>moderator</value>
      <value>participant</value>
      <value>visitor</value>
      <option label='Moderator'><value>moderator</value></option>
      <option label='Participant'><value>participant</value></option
          >
      <option label='Visitor'><value>visitor</value></option>
    </field>
    <field
        label='Roles␣and␣Affiliations␣that␣May␣Retrieve␣Member␣List'
        type='list-multi'
        var='muc#roomconfig_getmemberlist'>
      <value>moderator</value>
      <value>participant</value>
      <value>visitor</value>
      <option label='Moderator'><value>moderator</value></option>
      <option label='Participant'><value>participant</value></option
          >
      <option label='Visitor'><value>visitor</value></option>
    </field>
    <field
        label='Make␣Room␣Publicly␣Searchable?'
        type='boolean'
        var='muc#roomconfig_publicroom'>
      <value>1</value>
    </field>
    <field
        label='Make␣Room␣Persistent?'
        type='boolean'
        var='muc#roomconfig_persistentroom'>
      <value>0</value>
    </field>
    <field
        label='Make␣Room␣Moderated?'
        type='boolean'
        var='muc#roomconfig_moderatedroom'>
      <value>0</value>
    </field>
    <field
        label='Make␣Room␣Members-Only?'
```

```
      type='boolean'
      var='muc#roomconfig_membersonly'>
    <value>0</value>
  </field>
  <field
      label='Password Required to Enter?'
      type='boolean'
      var='muc#roomconfig_passwordprotectedroom'>
    <value>0</value>
  </field>
  <field type='fixed'>
    <value>
      If a password is required to enter this room,
      you must specify the password below.
    </value>
  </field>
  <field
      label='Password'
      type='text-private'
      var='muc#roomconfig_roomsecret'/>
  <field
      label='Who May Discover Real JIDs?'
      type='list-single'
      var='muc#roomconfig_whois'>
    <option label='Moderators Only'>
      <value>moderators</value>
    </option>
    <option label='Anyone'>
      <value>anyone</value>
    </option>
  </field>
  <field
      label='Maximum Number of History Messages Returned by Room'
      type='text-single'
      var='muc#maxhistoryfetch'>
    <value>50</value>
  </field>
  <field type='fixed'>
    <value>
      You may specify additional people who have
      admin status in the room. Please
      provide one Jabber ID per line.
    </value>
  </field>
  <field
      label='Room Admins'
      type='jid-multi'
      var='muc#roomconfig_roomadmins'/>
  <field type='fixed'>
```

```
        <value>
          You may specify additional owners for this
          room. Please provide one Jabber ID per line.
        </value>
      </field>
      <field
          label='Room␣Owners'
          type='jid-multi'
          var='muc#roomconfig_roomowners'/>
    </x>
  </query>
</iq>
```

Note: The _whois configuration option specifies whether the room is non-anonymous (a value of "anyone") or semi-anonymous (a value of "moderators").
A value of "none" was used in the past for fully-anonymous rooms, but it has been depreciated.
If there are no configuration options available, the service MUST return an empty query element to the room owner:

Listing 158: Service Informs Owner that No Configuration is Possible

```
<iq from='coven@chat.shakespeare.lit'
    id='create1'
    to='crone1@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#owner'/>
</iq>
```

The room owner SHOULD then fill out the form and submit it to the service.

Listing 159: Owner Submits Configuration Form

```
<iq from='crone1@shakespeare.lit/desktop'
    id='create2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='FORM_TYPE'>
        <value>http://jabber.org/protocol/muc#roomconfig</value>
      </field>
      <field var='muc#roomconfig_roomname'>
        <value>A Dark Cave</value>
      </field>
      <field var='muc#roomconfig_roomdesc'>
        <value>The place for all good witches!</value>
      </field>
      <field var='muc#roomconfig_enablelogging'>
        <value>0</value>
```

```
      </field>
      <field var='muc#roomconfig_changesubject'>
        <value>1</value>
      </field>
      <field var='muc#roomconfig_allowinvites'>
        <value>0</value>
      </field>
      <field var='muc#roomconfig_allowpm'>
        <value>anyone</value>
      </field>
      <field var='muc#roomconfig_maxusers'>
        <value>10</value>
      </field>
      <field var='muc#roomconfig_publicroom'>
        <value>0</value>
      </field>
      <field var='muc#roomconfig_persistentroom'>
        <value>0</value>
      </field>
      <field var='muc#roomconfig_moderatedroom'>
        <value>0</value>
      </field>
      <field var='muc#roomconfig_membersonly'>
        <value>0</value>
      </field>
      <field var='muc#roomconfig_passwordprotectedroom'>
        <value>1</value>
      </field>
      <field var='muc#roomconfig_roomsecret'>
        <value>cauldronburn</value>
      </field>
      <field var='muc#roomconfig_whois'>
        <value>moderators</value>
      </field>
      <field var='muc#maxhistoryfetch'>
        <value>50</value>
      </field>
      <field var='muc#roomconfig_roomadmins'>
        <value>wiccarocks@shakespeare.lit</value>
        <value>hecate@shakespeare.lit</value>
      </field>
    </x>
  </query>
</iq>
```

If room creation is successful, the service MUST inform the new room owner of success:

Listing 160: Service Informs New Room Owner of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='create2'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

If the room creation fails because the specified room configuration options violate one or more service policies (e.g., because the password for a password-protected room is blank), the service MUST return a <not-acceptable/> error.

Listing 161: Service Informs Owner that Requested Configuration Options Are Unacceptable

```
<iq from='coven@chat.shakespeare.lit'
    id='create2'
    to='crone1@shakespeare.lit/desktop'
    type='error'>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Alternatively, the room owner MAY cancel the configuration process:

Listing 162: Owner Cancels Initial Configuration

```
<iq from='crone1@shakespeare.lit/desktop'
    id='create2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <x xmlns='jabber:x:data' type='cancel'/>
  </query>
</iq>
```

If the room owner cancels the initial configuration, the service MUST destroy the room, making sure to send unavailable presence to the room owner (see the "Destroying a Room" use case for protocol details).
If the room owner becomes unavailable for any reason before submitting the form (e.g., a lost connection), the service will receive a presence stanza of type "unavailable" from the owner to the owner's <room@service/nick>. The service MUST then destroy the room, sending a presence stanza of type "unavailable" from the room to the owner including a <destroy/> element and reason (if provided) as defined in the Destroying a Room section of this document.

10.2 Subsequent Room Configuration

At any time after specifying the initial configuration of the room, a room owner might want to change the configuration. In order to initiate this process, a room owner requests a new configuration form from the room by sending an IQ get to <room@service> containing an empty

<query/> element qualified by the 'http://jabber.org/protocol/muc#owner' namespace.

Listing 163: Owner Requests Configuration Form

```
<iq from='crone1@shakespeare.lit/desktop'
    id='config1'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/muc#owner'/>
</iq>
```

If the <user@host> of the 'from' address does not match the bare JID of a room owner, the service MUST return a <forbidden/> error to the sender:

Listing 164: Service Denies Configuration Access to Non-Owner

```
<iq from='coven@chat.shakespeare.lit'
    id='configures'
    to='wiccarocks@shakespeare.lit/laptop'
    type='error'>
  <query xmlns='http://jabber.org/protocol/muc#owner'/>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Otherwise, the service MUST send a configuration form to the room owner with the current options set as defaults:

Listing 165: Service Sends Configuration Form to Owner

```
<iq from='coven@chat.shakespeare.lit'
    id='config1'
    to='crone1@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <x xmlns='jabber:x:data' type='form'>
      <title>Configuration for "coven" Room</title>
      <instructions>
        Complete this form to modify the
        configuration of your room.
      </instructions>
      <field
          type='hidden'
          var='FORM_TYPE'>
        <value>http://jabber.org/protocol/muc#roomconfig</value>
      </field>
      <field
          label='Natural-Language␣Room␣Name'
```

```
      type='text-single'
      var='muc#roomconfig_roomname'>
    <value>A Dark Cave</value>
  </field>
  <field
      label='Short Description of Room'
      type='text-single'
      var='muc#roomconfig_roomdesc'>
    <value>The place for all good witches!</value>
  </field>
  <field
      label='Enable Public Logging?'
      type='boolean'
      var='muc#roomconfig_enablelogging'>
    <value>0</value>
  </field>
  <field
      label='Allow Occupants to Change Subject?'
      type='boolean'
      var='muc#roomconfig_changesubject'>
    <value>0</value>
  </field>
  <field
      label='Allow Occupants to Invite Others?'
      type='boolean'
      var='muc#roomconfig_allowinvites'>
    <value>0</value>
  </field>
  <field
      label='Who Can Send Private Messages?'
      type='list-single'
      var='muc#roomconfig_allowpm'>
    <value>anyone</value>
    <option label='Anyone'>
      <value>anyone</value>
    </option>
    <option label='Anyone with Voice'>
      <value>participants</value>
      </option>
    <option label='Moderators Only'>
      <value>moderators</value>
    </option>
    <option label='Nobody'>
      <value>none</value>
    </option>
  </field>
  <field
      label='Maximum Number of Occupants'
      type='list-single'
```

```
      var='muc#roomconfig_maxusers'>
    <value>10</value>
    <option label='10'><value>10</value></option>
    <option label='20'><value>20</value></option>
    <option label='30'><value>30</value></option>
    <option label='50'><value>50</value></option>
    <option label='100'><value>100</value></option>
    <option label='None'><value>none</value></option>
  </field>
  <field
      label='Roles␣for␣which␣Presence␣is␣Broadcasted'
      type='list-multi'
      var='muc#roomconfig_presencebroadcast'>
    <value>moderator</value>
    <value>participant</value>
    <value>visitor</value>
    <option label='Moderator'><value>moderator</value></option>
    <option label='Participant'><value>participant</value></option
       >
    <option label='Visitor'><value>visitor</value></option>
  </field>
  <field
      label='Roles␣and␣Affiliations␣that␣May␣Retrieve␣Member␣List'
      type='list-multi'
      var='muc#roomconfig_getmemberlist'>
    <value>moderator</value>
    <value>participant</value>
    <value>visitor</value>
    <option label='Moderator'><value>moderator</value></option>
    <option label='Participant'><value>participant</value></option
       >
    <option label='Visitor'><value>visitor</value></option>
  </field>
  <field
      label='Make␣Room␣Publicly␣Searchable?'
      type='boolean'
      var='muc#roomconfig_publicroom'>
    <value>0</value>
  </field>
  <field
      label='Make␣Room␣Persistent?'
      type='boolean'
      var='muc#roomconfig_persistentroom'>
    <value>0</value>
  </field>
  <field
      label='Make␣Room␣Moderated?'
      type='boolean'
      var='muc#roomconfig_moderatedroom'>
```

```
      <value>0</value>
    </field>
    <field
        label='Make␣Room␣Members␣Only?'
        type='boolean'
        var='muc#roomconfig_membersonly'>
      <value>0</value>
    </field>
    <field
        label='Password␣Required␣for␣Entry?'
        type='boolean'
        var='muc#roomconfig_passwordprotectedroom'>
      <value>1</value>
    </field>
    <field type='fixed'>
      <value>
        If a password is required to enter this room,
        you must specify the password below.
      </value>
    </field>
    <field
        label='Password'
        type='text-private'
        var='muc#roomconfig_roomsecret'>
      <value>cauldronburn</value>
    </field>
    <field
        label='Who␣May␣Discover␣Real␣JIDs?'
        type='list-single'
        var='muc#roomconfig_whois'>
      <value>moderators</value>
      <option label='Moderators␣Only'>
        <value>moderators</value>
      </option>
      <option label='Anyone'>
        <value>anyone</value>
      </option>
    </field>
    <field
        label='Maximum␣Number␣of␣History␣Messages␣Returned␣by␣Room'
        type='text-single'
        var='muc#maxhistoryfetch'>
      <value>50</value>
    </field>
    <field type='fixed'>
      <value>
        You may specify additional people who have
        admin status in the room. Please
        provide one Jabber ID per line.
```

```
        </value>
      </field>
      <field
          label='Room Admins'
          type='jid-multi'
          var='muc#roomconfig_roomadmins'>
        <value>wiccarocks@shakespeare.lit</value>
        <value>hecate@shakespeare.lit</value>
      </field>
      <field type='fixed'>
        <value>
          You may specify additional owners for this
          room. Please provide one Jabber ID per line.
        </value>
      </field>
      <field
          label='Room Owners'
          type='jid-multi'
          var='muc#roomconfig_roomowners'/>
    </x>
  </query>
</iq>
```

If there are no configuration options available, the service MUST return an empty query element to the room owner as shown in the previous use case.
The room owner then submits the form with updated configuration information. (Example not shown.)
Alternatively, the room owner MAY cancel the configuration process:

Listing 166: Owner Cancels Subsequent Configuration

```
<iq from='crone1@shakespeare.lit/desktop'
    id='config2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <x xmlns='jabber:x:data' type='cancel'/>
  </query>
</iq>
```

If the room owner cancels the subsequent configuration, the service MUST leave the configuration of the room as it was before the room owner initiated the subsequent configuration process.
If as a result of a change in the room configuration a room admin loses admin status while in the room, the room MUST send updated presence for that individual to all occupants, denoting the change in status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "member" or "none" and the 'role' attribute set

to a value of "participant" or "visitor" as appropriate for the affiliation level and the room type:

Listing 167: Service Notes Loss of Admin Affiliation

```
<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='wiccarocks@shakespeare.lit/laptop'
          role='participant'/>
  </x>
</presence>

[ ... ]
```

If as a result of a change in the room configuration a user gains admin status while in the room, the room MUST send updated presence for that individual to all occupants, denoting the change in status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "admin" and the 'role' attribute set to a value of "moderator":

Listing 168: Service Notes Gain of Admin Affiliation to All Users

```
<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='admin'
          jid='wiccarocks@shakespeare.lit/laptop'
          role='moderator'/>
  </x>
</presence>

[ ... ]
```

If as a result of a change in the room configuration a room owner loses owner status while that owner is in the room, the room MUST send updated presence for that individual to all occupants, denoting the change in status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "admin" and the 'role' attribute set to an appropriate value given the affiliation and room type ("moderator" is recommended).

Listing 169: Service Notes Loss of Owner Affiliation

```
<presence
    from='coven@chat.shakespeare.lit/secondwitch'
```

```
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='admin'
          jid='wiccarocks@shakespeare.lit/laptop'
          role='moderator'/>
  </x>
</presence>

[ ... ]
```

A service MUST NOT allow an owner to revoke his or her own owner status if there are no other owners; if an owner attempts to do this, the service MUST return a <conflict/> error to the owner. However, a service SHOULD allow an owner to revoke his or her own owner status if there are other owners.
If as a result of a change in the room configuration a user gains owner status while in the room, the room MUST send updated presence for that individual to all occupants, denoting the change in status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "owner" and the 'role' attribute set to an appropriate value given the affiliation and room type ("moderator" is recommended).

Listing 170: Service Notes Gain of Owner Affiliation to All Users

```
<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='owner'
          jid='wiccarocks@shakespeare.lit/laptop'
          role='moderator'/>
  </x>
</presence>

[ ... ]
```

If as a result of a change in the room configuration the room type is changed to members-only but there are non-members in the room, the service MUST remove any non-members from the room and include a status code of 322 in the presence unavailable stanzas sent to those users as well as any remaining occupants.

10.2.1 Notification of Configuration Changes

A room MUST send notification to all occupants when the room configuration changes in a way that has an impact on the privacy or security profile of the room. This notification shall consist of a <message/> stanza containing an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace, which shall contain only a <status/>

element with an appropriate value for the 'code' attribute. Here is an example:

Listing 171: Configuration Status Code

```
<message from='coven@chat.shakespeare.lit'
         id='80349046-F26A-44F3-A7A6-54825064DD9E'
         to='crone1@shakespeare.lit/desktop'
         type='groupchat'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <status code='170'/>
  </x>
</message>
```

The codes to be generated as a result of a privacy-related change in room configuration are as follows:

- If room logging is now enabled, status code 170.
- If room logging is now disabled, status code 171.
- If the room is now non-anonymous, status code 172.
- If the room is now semi-anonymous, status code 173.

For any other configuration change, the room SHOULD send status code 104 so that interested occupants can retrieve the updated room configuration if desired.

10.3 Granting Owner Status

If allowed by an implementation, an owner MAY grant owner status to another user; this is done by changing the user's affiliation to "owner":

Listing 172: Owner Grants Owner Status

```
<iq from='crone1@shakespeare.lit/desktop'
    id='owner1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='owner'
          jid='hecate@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 173: Owner Grants Owner Status (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='owner1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='owner'
          jid='hecate@shakespeare.lit'>
      <reason>A worthy witch indeed!</reason>
    </item>
  </query>
</iq>
```

As affiliations are granted, revoked, and maintained based on the user's bare JID, the requesting entity SHOULD use the bare JID of the user in the request. When processing a request that identifies a user by its full JID, a service SHOULD use the bare JID representation.
If the <user@host> of the 'from' address does not match the bare JID of a room owner, the service MUST return a <forbidden/> error to the sender.
Otherwise, the service MUST add the user to the owner list and then inform the owner of success:

Listing 174: Service Informs Owner of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='owner1'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

If the user is in the room, the service MUST then send updated presence from this individual to all occupants, indicating the granting of owner status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "owner" and the 'role' attribute set to an appropriate value given the affiliation and room type ("moderator" is recommended).

Listing 175: Service Sends Notice of Owner Status to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/hecate'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='owner'
          jid='hecate@shakespeare.lit'
          role='moderator'/>
  </x>
</presence>

[ ... ]
```

If the user is not in the room, the service MAY send a message from the room itself to the room occupants, indicating the granting of owner status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "owner".

Listing 176: Service Sends Notice of Owner Status to All Occupants

```
<message
    from='chat.shakespeare.lit'
    id='22B0F570-526A-4F22-BDE3-52EC3BB18371'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='member'
          jid='hecate@shakespeare.lit'
          role='none'/>
  </x>
</message>

[ ... ]
```

10.4 Revoking Owner Status

An implementation MAY allow an owner to revoke another user's owner status; this is done by changing the user's affiliation to something other than "owner":

Listing 177: Owner Revokes Owner Status

```
<iq from='crone1@shakespeare.lit/desktop'
    id='owner2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='admin'
          jid='hecate@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 178: Owner Revokes Owner Status (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='owner2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='admin'
```

```
          jid='hecate@shakespeare.lit'>
      <reason>Not so worthy after all!</reason>
    </item>
  </query>
</iq>
```

As affiliations are granted, revoked, and maintained based on the user's bare JID, the requesting entity SHOULD use the bare JID of the user in the request. When processing a request that identifies a user by its full JID, a service SHOULD use the bare JID representation.
If the <user@host> of the 'from' address does not match the bare JID of a room owner, the service MUST return a <forbidden/> error to the sender.
A service MUST NOT allow an owner to revoke his or her own owner status if there are no other owners; if an owner attempts to do this, the service MUST return a <conflict/> error to the owner. However, a service SHOULD allow an owner to revoke his or her own owner status if there are other owners.
If an implementation does not allow one owner to revoke another user's owner status, the implementation MUST return a <not-authorized/> error to the owner who made the request.
Note: Allowing an owner to remove another user's owner status can compromise the control model for room management; therefore this feature is OPTIONAL, and implementations are encouraged to support owner removal through an interface that is open only to individuals with service-wide admin status.
In all other cases, the service MUST remove the user from the owner list and then inform the owner of success:

Listing 179: Service Informs Owner of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='owner2'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

If the user is in the room, the service MUST then send updated presence from this individual to all occupants, indicating the loss of owner status by sending a presence element that contains an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value other than "owner" and the 'role' attribute set to an appropriate value:

Listing 180: Service Notes Loss of Owner Affiliation

```
<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='admin'
          jid='hecate@shakespeare.lit'
          role='moderator'/>
```

```
  </x>
</presence>

[ ... ]
```

10.5 Modifying the Owner List

If allowed by an implementation, a room owner might want to modify the owner list. To do so, the owner first requests the owner list by querying the room for all users with an affiliation of 'owner'.

Listing 181: Owner Requests Owner List

```
<iq from='bard@shakespeare.lit/globe'
    id='owner3'
    to='coven@chat.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='owner'/>
  </query>
</iq>
```

If the <user@host> of the 'from' address does not match the bare JID of a room owner, the service MUST return a <forbidden/> error to the sender in semi-anonymous rooms. In non-anonymous rooms, the service MAY process the request.
Otherwise, the service MUST then return the owner list to the owner; each item MUST include the 'affiliation' and 'jid' attributes and MAY include the 'nick' and 'role' attributes for any owner that is currently an occupant:

Listing 182: Service Sends Owner List to Owner

```
<iq from='coven@chat.shakespeare.lit'
    id='owner3'
    to='bard@shakespeare.lit/globe'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='owner'
          jid='crone1@shakespeare.lit'/>
  </query>
</iq>
```

The owner can then modify the owner list if desired. In order to do so, the owner MUST send the changed items (i.e., only the "delta") back to the service; [36] each item MUST include the 'affiliation' and 'jid' attributes but SHOULD NOT include the 'nick' attribute and MUST NOT

[36] This is different from the behavior of room configuration, wherein the "muc#roomconfig_roomowners" field specifies the full list of room owners, not the delta.

include the 'role' attribute (which is used to manage roles such as participant rather than affiliations such as owner). As affiliations are granted, revoked, and maintained based on the user's bare JID, the requesting entity SHOULD use the bare JID of users in the request. When processing a request that identifies a user by its full JID, a service SHOULD use the bare JID representation.

Listing 183: Owner Sends Modified Owner List to Service

```
<iq from='bard@shakespeare.lit/globe'
    id='owner4'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='owner'
          jid='hecate@shakespeare.lit'/>
  </query>
</iq>
```

Only owners shall be allowed to modify the owner list. If a non-owner attempts to modify the owner list, the service MUST deny the request and return a <forbidden/> error to the sender:

Listing 184: Service Returns Error on Attempt by Non-Owner to Modify Owner List

```
<iq from='coven@chat.shakespeare.lit'
    id='ownertest'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

A service MUST NOT allow an owner to revoke his or her own owner status if there are no other owners; if an owner attempts to do this, the service MUST return a <conflict/> error to the owner. However, a service SHOULD allow an owner to revoke his or her own owner status if there are other owners.
In all other cases, the service MUST modify owner list and then inform the owner of success:

Listing 185: Service Informs Owner of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='owner4'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST also send presence notifications related to any affiliation changes that result from modifying the owner list as previously described.

10.6 Granting Admin Status

An owner can grant admin status to a member or an unaffiliated user; this is done by changing the user's affiliation to "admin":

Listing 186: Owner Grants Admin Privileges

```
<iq from='crone1@shakespeare.lit/desktop'
    id='admin1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='admin'
          jid='wiccarocks@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 187: Owner Grants Admin Privileges (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='admin1'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='admin'
          jid='wiccarocks@shakespeare.lit'>
      <reason>A worthy witch indeed!</reason>
    </item>
  </query>
</iq>
```

As affiliations are granted, revoked, and maintained based on the user's bare JID, the requesting entity SHOULD use the bare JID of the user in the request. When processing a request that identifies a user by its full JID, a service SHOULD use the bare JID representation.
If the <user@host> of the 'from' address does not match the bare JID of a room owner, the service MUST return a <forbidden/> error to the sender.
Otherwise, the service MUST add the user to the admin list and then inform the owner of success:

Listing 188: Service Informs Owner of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='admin1'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

If the user is in the room, the service MUST then send updated presence from this individual to all occupants, indicating the granting of admin status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "admin" and the 'role' attribute set to an appropriate value given the affiliation and room type (typically "moderator").

Listing 189: Service Sends Notice of Admin Status to All Occupants

```
<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='admin'
          jid='wiccarocks@shakespeare.lit'
          role='moderator'/>
  </x>
</presence>

[ ... ]
```

If the user is not in the room, the service MAY send a message from the room itself to the room occupants, indicating the granting of admin status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "admin".

Listing 190: Service Sends Notice of Admin Status to All Occupants

```
<message
    from='chat.shakespeare.lit'
    id='C75B919A-30B3-4233-AE89-6E9834E26929'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='admin'
          jid='wiccarocks@shakespeare.lit'
          role='none'/>
  </x>
</message>

[ ... ]
```

10.7 Revoking Admin Status

An owner might want to revoke a user's admin status; this is done by changing the user's affiliation to something other than "admin" or "owner" (typically to "member" in a members-only room or to "none" in other types of room).

Listing 191: Owner Revokes Admin Status

```
<iq from='crone1@shakespeare.lit/desktop'
    id='admin2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='none'
          jid='wiccarocks@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Listing 192: Owner Revokes Admin Status (With a Reason)

```
<iq from='crone1@shakespeare.lit/desktop'
    id='admin2'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='none'
          jid='wiccarocks@shakespeare.lit'>
      <reason>Not so worthy after all!</reason>
    </item>
  </query>
</iq>
```

As affiliations are granted, revoked, and maintained based on the user's bare JID, the requesting entity SHOULD use the bare JID of the user in the request. When processing a request that identifies a user by its full JID, a service SHOULD use the bare JID representation.
If the <user@host> of the 'from' address does not match the bare JID of a room owner, the service MUST return a <forbidden/> error to the sender.
Otherwise, the service MUST remove the user from the admin list and then inform the owner of success:

Listing 193: Service Informs Owner of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='admin2'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

If the user is in the room, the service MUST then send updated presence from this individual to all occupants, indicating the loss of admin status by sending a presence element that contains an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value other than "admin" or "owner" and the 'role' attribute set to an appropriate value given the affiliation

level and the room type (typically "participant").

Listing 194: Service Notes Loss of Admin Affiliation

```
<presence
    from='coven@chat.shakespeare.lit/secondwitch'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none'
          jid='wiccarocks@shakespeare.lit'
          role='participant'/>
  </x>
</presence>

[ ... ]
```

If the user is not in the room, the service MAY send a message from the room itself to the room occupants, indicating the loss of admin status by including an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value other than "admin".

Listing 195: Service Notes Loss of Admin Affiliation

```
<message
    from='chat.shakespeare.lit'
    id='2CF9013B-E8A8-42A1-9633-85AD7CA12F40'
    to='crone1@shakespeare.lit/desktop'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none'
          jid='wiccarocks@shakespeare.lit'
          role='none'/>
  </x>
</message>

[ ... ]
```

10.8 Modifying the Admin List

A room owner might want to modify the admin list. To do so, the owner first requests the admin list by querying the room for all users with an affiliation of 'admin'.

Listing 196: Owner Requests Admin List

```
<iq from='bard@shakespeare.lit/desktop'
    id='admin3'
    to='coven@chat.shakespeare.lit'
    type='get'>
```

```
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='admin'/>
  </query>
</iq>
```

If the <user@host> of the 'from' address does not match the bare JID of a room owner, the service MUST return a <forbidden/> error to the sender in semi-anonymous rooms. In non-anonymous rooms, the service MAY process the request.
Otherwise, the service MUST then return the admin list to the owner; each item MUST include the 'affiliation' and 'jid' attributes and MAY include the 'nick' and 'role' attributes for any admin that is currently an occupant:

Listing 197: Service Sends Admin List to Owner

```
<iq from='coven@chat.shakespeare.lit'
    id='admin3'
    to='bard@shakespeare.lit/globe'
    type='result'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='admin'
          jid='wiccarocks@shakespeare.lit'
          nick='secondwitch'/>
    <item affiliation='admin'
          jid='hag66@shakespeare.lit'/>
  </query>
</iq>
```

The owner can then modify the admin list if desired. In order to do so, the owner MUST send the changed items (i.e., only the "delta") back to the service; [37] each item MUST include the 'affiliation' attribute (normally set to a value of "admin" or "none") and 'jid' attribute but SHOULD NOT include the 'nick' attribute and MUST NOT include the 'role' attribute (which is used to manage roles such as participant rather than affiliations such as owner). As affiliations are granted, revoked, and maintained based on the user's bare JID, the requesting entity SHOULD use the bare JID of users in the request. When processing a request that identifies a user by its full JID, a service SHOULD use the bare JID representation.

Listing 198: Owner Sends Modified Admin List to Service

```
<iq from='bard@shakespeare.lit/globe'
    id='admin4'
    to='coven@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#admin'>
    <item affiliation='none'
          jid='hag66@shakespeare.lit'>
```

[37] This is different from the behavior of room configuration, wherein the "muc#roomconfig_roomadmins" field specifies the full list of room admins, not the delta.

```
    </item>
    <item affiliation='admin'
          jid='hecate@shakespeare.lit'>
    </item>
  </query>
</iq>
```

Only owners shall be allowed to modify the admin list. If a non-owner attempts to modify the admin list, the service MUST deny the request and return a <forbidden/> error to the sender.

Listing 199: Service Returns Error on Attempt by Non-Owner to Modify Admin List

```
<iq from='coven@chat.shakespeare.lit'
    id='admintest'
    to='hag66@shakespeare.lit/pda'
    type='error'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Otherwise, the service MUST modify the admin list and then inform the owner of success:

Listing 200: Service Informs Owner of Success

```
<iq from='coven@chat.shakespeare.lit'
    id='admin4'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST also send presence notifications related to any affiliation changes that result from modifying the admin list as previously described.

10.9 Destroying a Room

A room owner MUST be able to destroy a room, especially if the room is persistent. The workflow is as follows:

1. The room owner requests that the room be destroyed, optionally specifying a reason and an alternate venue.

2. The room removes all users from the room (including appropriate information about the alternate location and the reason for being removed) and destroys the room, even if it was defined as persistent.

Other than the foregoing, this document does not specify what (if anything) a MUC service implementation shall do as a result of a room destruction request. For example, if the room was defined as persistent, an implementation MAY choose to lock the room ID so that it cannot be re-used, redirect enter requests to the alternate venue, or invite the current participants to the new room; however, such behavior is OPTIONAL.
In order to destroy a room, the room owner MUST send an IQ set to the address of the room to be destroyed. The <iq/> stanza shall contain a <query/> element qualified by the 'http://jabber.org/protocol/muc#owner' namespace, which in turn shall contain a <destroy/> element. The address of the alternate venue MAY be provided as the value of the <destroy/> element's 'jid' attribute. A password for the alternate venue MAY be provided as the XML character data of a <password/> child element of the <destroy/> element. The reason for the room destruction MAY be provided as the XML character data of a <reason/> child element of the <destroy/> element.
The following examples illustrate the protocol elements to be sent and received:

Listing 201: Owner Submits Room Destruction Request

```
<iq from='crone1@shakespeare.lit/desktop'
    id='begone'
    to='heath@chat.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
    <destroy jid='coven@chat.shakespeare.lit'>
      <reason>Macbeth doth come.</reason>
    </destroy>
  </query>
</iq>
```

The service is responsible for removing all the occupants. It SHOULD NOT broadcast presence stanzas of type "unavailable" from all occupants, instead sending only one presence stanza of type "unavailable" to each occupant so that the user knows he or she has been removed from the room. The extended presence information of the stanza MUST include <destroy/> element. If extended information specifying the JID of an alternate location and/or the reason for the room destruction was provided by the room owner, the presence stanza MUST include that information.
Note: in older versions of this standard, the inclusion of a <destroy/> element was not explicitly defined to be mandatory (when no alternate location or reason are provided by the room owner). Implementations therefore cannot reliably depend on the existence of this element to identify a room destruction event.

Listing 202: Service Removes Each Occupant

```
<presence
    from='heath@chat.shakespeare.lit/firstwitch'
    to='crone1@shakespeare.lit/desktop'
    type='unavailable'>
```

```
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none'/>
    <destroy jid='coven@chat.shakespeare.lit'>
      <reason>Macbeth doth come.</reason>
    </destroy>
  </x>
</presence>

<presence
    from='heath@chat.shakespeare.lit/secondwitch'
    to='wiccarocks@shakespeare.lit/laptop'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none'/>
    <destroy jid='coven@chat.shakespeare.lit'>
      <reason>Macbeth doth come.</reason>
    </destroy>
  </x>
</presence>

<presence
    from='heath@chat.shakespeare.lit/thirdwitch'
    to='hag66@shakespeare.lit/pda'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none'/>
    <destroy jid='coven@chat.shakespeare.lit'>
      <reason>Macbeth doth come.</reason>
    </destroy>
  </x>
</presence>
```

Listing 203: Service Informs Owner of Successful Destruction

```
<iq from='heath@chat.shakespeare.lit'
    id='begone'
    to='crone1@shakespeare.lit/desktop'
    type='result'/>
```

If the <user@host> of the 'from' address received on a destroy request does not match the bare JID of a room owner, the service MUST return a <forbidden/> error to the sender:

Listing 204: Service Denies Destroy Request Submitted by Non-Owner

```
<iq from='heath@chat.shakespeare.lit'
    id='destroytest'
    to='wiccarocks@shakespeare.lit/laptop'
    type='error'>
  <query xmlns='http://jabber.org/protocol/muc#owner'>
```

```
    <destroy jid='coven@chat.shakespeare.lit'>
      <reason>Macbeth doth come.</reason>
    </destroy>
  </query>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

11 Service Use Cases

11.1 Service removes user because of error response

A MUC service MAY support adding the 333 status code to presences when a user gets removed by the service due to a technical problem (e.g. s2s link failure). This can, for example, be used as a hint for clients to distinguish between an occupant getting disconnected and them intentionally leaving the room.
If a MUC service supports this OPTIONAL feature, it MUST include the 333 status code in the resulting presence:

Listing 205: MUC service removes user because of error

```
<presence
    from='harfleur@chat.shakespeare.lit/pistol'
    to='pistol@shakespeare.lit/harfleur'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none' />
    <status code='110'/>
    <status code='333'/>
  </x>
</presence>
```

The status code MUST also be included in presences sent to other occupants:

Listing 206: MUC service informs other occupants of removal because of an error

```
<presence
    from='harfleur@chat.shakespeare.lit/pistol'
    to='gower@shakespeare.lit/cell'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none'/>
    <status code='333'/>
  </x>
</presence>
```

Note: Some server implementations additionally include a 307 status code (signifying a 'kick', i.e. a forced ejection from the room). This is generally not advisable, as these types of disconnects may be frequent in the presence of poor network conditions and they are not linked to any user (e.g. moderator) action that the 307 code usually indicates. It is therefore recommended for the client to ignore the 307 code if a 333 status code is present.

11.2 Service removes user because of service shut down

When a MUC service shuts downs, it SHOULD inform its participant by sending presences containing the 332 status code

Listing 207: MUC service removes user because of service shutdown

```
<presence
    from='harfleur@chat.shakespeare.lit/pistol'
    to='pistol@shakespeare.lit/client'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none' />
    <status code='110'/>
    <status code='332'/>
  </x>
</presence>
<presence
    from='harfleur@chat.shakespeare.lit/othello'
    to='othello@shakespeare.lit/client'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <item affiliation='none' role='none' />
    <status code='110'/>
    <status code='332'/>
  </x>
</presence>
```

12 Status Codes

Multi-User Chat uses a <status/> element (specifically, the 'code' attribute of the <status/> element) to communicate information about a user's status in a room. Over time, the number of status codes has grown quite large, and new status codes continue to be requested of the author. Therefore, these codes are now documented in a registry maintained by the XMPP Registrar. For details, refer to the Status Codes Registry section of this document.
Note: In general, MUC status codes tend to follow the "philosophy" of status codes that is

implicit in RFC 2616 [38] and RFC 1893 [39] (1xx codes are informational, 2xx codes specify that it is fine to continue, 3xx codes specify redirects such as being kicked or banned, x3x codes refer to system status, x7x codes refer to security or policy matters, etc.).
Note: If the MUC protocol were being designed today, it would specify a more flexible, XML-friendly approach rather than hardcoded status numbers; however, at this point the pain of changing the status reporting system would be greater than the benefit of doing so, which is why the status code numbers remain in use. A future version of this document may define a more XMPP-like approach to status conditions, retaining the code numbers but supplementing them with more descriptive child elements as is done in RFC 6120 [40].

13 Internationalization Considerations

As specified in RFC 6120 [41], XMPP entities (including MUC rooms and MUC services) SHOULD respect the value of the 'xml:lang' attribute provided with any given stanza. However, simultaneous translation of groupchat messages is out of scope for this document (see Language Translation (XEP-0171) [42]).
The status and error codes defined herein enable a client implementation to present a localized interface; however, definition of the localized text strings for any given language community is out of scope for this document.
Although the labels for various data form fields are shown here in English, MUC clients SHOULD present localized text for these fields rather than the English text.
Nicknames can contain virtually any Unicode character. This introduces the possibility of nick spoofing; see RFC 6122 [43] for a description of related security considerations.

14 Security Considerations

14.1 User Authentication and Authorization

No room entrance authentication or authorization method more secure than cleartext passwords is defined or required by this document. Although the risks involved can mitigated somewhat by the use of channel encryption and strong authentication via TLS and SASL as described in RFC 6120 [44], an entity that joins a room has no way of knowing if its complete communication channel to the room is encrypted (thereby protecting the plaintext password). A future specification might define an XMPP profile of SASL for use with MUC, but

[38] RFC 2616: Hypertext Transport Protocol -- HTTP/1.1 <http://tools.ietf.org/html/rfc2616>.
[39] RFC 1893: Enhanced Mail System Status Codes <http://tools.ietf.org/html/rfc1893>.
[40] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[41] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[42] XEP-0171: Language Translation <https://xmpp.org/extensions/xep-0171.html>.
[43] RFC 6122: Extensible Messaging and Presence Protocol (XMPP): Address Format <http://tools.ietf.org/html/rfc6122>.
[44] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

currently there is no such specification.

14.2 End-to-End Encryption

No end-to-end message or session encryption method is specified herein. Users SHOULD NOT trust a service to keep secret any text sent through a room. A future specification might define a method for end-to-end encryption of MUC traffic, but currently there is no such specification.

14.3 Privacy

Depending on room configuration, a room might publicly log all discussions held in the room. A service MUST warn the user that the room is publicly logged by returning a status code of "170" with the user's initial presence, and the user's client MUST warn the user if the room discussion is logged (a user's client SHOULD also query the room for its configuration prior to allowing the user to enter in order to "pre-discover" whether the room is logged). A client MUST also warn the user if the room's configuration is subsequently modified to allow room logging (which the client will discover when the room sends status code 170).
Note: In-room history is different from public room logging, and naturally a room cannot effectively prevent occupants from separately maintaining their own room logs, which may become public; users SHOULD exercise due caution and consider any room discussions to be effectively public.

14.4 Information Leaks

The "roominfo" data form used in extended service discovery can result in information leaks, e.g., the current discussion topic (via the "roominfo_subject" field). The same is true of service discovery items (disco#items) requests from outside the room (which could be used to discover the list of room occupants).
Implementations and deployments are advised to carefully consider the possibility that this information might be leaked, and to turn off information sharing by default for sensitive data.

14.5 Anonymity

Depending on room configuration, a room might expose each occupant's real JID to other occupants (if the room is non-anonymous). If real JIDs are exposed to all occupants in the room, the service MUST warn the user by returning a status code of "100" with the user's initial presence, and the user's client MUST warn the user (a user's client SHOULD also query the room for its configuration prior to allowing the user to enter in order to "pre-discover" whether real JIDs are exposed in the room). A client MUST also warn the user if the room's configuration is modified from semi-anonymous to non-anonymous (which the client will

discover when the room sends status code 172).

14.6 Denial of Service

Public MUC rooms can be subject to a number of attacks, most of which reduce to denial of service attacks. Such attacks include but are not limited to:

1. Stuffing the room with a large number of illegitimate occupants and therefore preventing legitimate users from joining the room.
2. Sending abusive messages and then leaving the room before a kick or ban can be applied; such abusive messages include but are not limited to large messages that prevent participants from following the conversation thread or room history, personal attacks on participants (especially room administrators and moderators), offensive text, and links to spam sites.
3. Making rapid and repeated presence changes.
4. Using long nicknames to route around lack of voice.
5. Abusing the room administrators or other room occupants.
6. Registering multiple nicknames across a service and therefore denying the use of those nicknames.
7. Mimicking another occupant's roomnick (e.g., by adding a space at the end or substituting visually similar characters), then sending messages from that roomnick in an effort to confuse the occupants.

These attacks can be mitigated but not completely prevented through the liberal use of administrative actions such as banning, the presence of automated room bots with admin status, implementation of intelligent content filtering, checking the IP addresses of connected users (not always possible in a distributed system), applying voice rules to presence as well as messaging, matching room nicks using more stringent rules than the Resourceprep profile of stringprep, etc. However, experience has shown that it is impossible to fully prevent attacks of this kind.
Public MUC services also can be subject to attacks, such as creating a large number of rooms on a service, leaving rooms in an unconfigured state, etc. Such service-level attacks can be mitigated by limiting the number of rooms that any given non-adminstrative user can own, deleting rooms if they remain in the unconfigured state for too long, etc.

14.7 Other Considerations

See Delayed Delivery (XEP-0203) [45] for security considerations regarding the inclusion and processing of delayed delivery notations.

15 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [46].

16 XMPP Registrar Considerations

The XMPP Registrar [47] includes the following information in its registries.

16.1 Protocol Namespaces

The XMPP Registrar includes the following MUC-related namespaces in its registry of protocol namespaces at <https://xmpp.org/registrar/namespaces.html>:

- http://jabber.org/protocol/muc
- http://jabber.org/protocol/muc#admin
- http://jabber.org/protocol/muc#owner
- http://jabber.org/protocol/muc#user

16.2 Service Discovery Category/Type

A Multi-User Chat service or room is identified by the "conference" category and the "text" type within Service Discovery.

[45] XEP-0203: Delayed Delivery <https://xmpp.org/extensions/xep-0203.html>.

[46] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

[47] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

16.3 Service Discovery Features

There are many features related to a MUC service or room that can be discovered by means of Service Discovery. The most fundamental of these is the 'http://jabber.org/protocol/muc' namespace. In addition, a MUC room SHOULD provide information about the specific room features it implements, such as password protection and room moderation.

```
<var>
  <name>http://jabber.org/protocol/muc#register</name>
  <desc>Support for the muc#register FORM_TYPE</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>http://jabber.org/protocol/muc#roomconfig</name>
  <desc>Support for the muc#roomconfig FORM_TYPE</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>http://jabber.org/protocol/muc#roominfo</name>
  <desc>Support for the muc#roominfo FORM_TYPE</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>http://jabber.org/protocol/muc#stable_id</name>
  <desc>This MUC will reflect the original message 'id' in 'groupchat'
      messages.</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_hidden</name>
  <desc>Hidden room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_membersonly</name>
  <desc>Members-only room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_moderated</name>
  <desc>Moderated room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_nonanonymous</name>
  <desc>Non-anonymous room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
```

```
  <name>muc_open</name>
  <desc>Open room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_passwordprotected</name>
  <desc>Password-protected room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_persistent</name>
  <desc>Persistent room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_public</name>
  <desc>Public room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_rooms</name>
  <desc>List of MUC rooms (each as a separate item)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_semianonymous</name>
  <desc>Semi-anonymous room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_temporary</name>
  <desc>Temporary room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_unmoderated</name>
  <desc>Unmoderated room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
<var>
  <name>muc_unsecured</name>
  <desc>Unsecured room in Multi-User Chat (MUC)</desc>
  <doc>XEP-0045</doc>
</var>
```

16.4 Well-Known Service Discovery Nodes

The well-known Service Discovery node 'http://jabber.org/protocol/muc#rooms' enables discovery of the rooms in which a user is an occupant.
The well-known Service Discovery node 'x-roomuser-item' enables a user to discover his or her registered roomnick from outside the room.
The well-known Service Discovery node 'http://jabber.org/protocol/muc#traffic' enables discovery of the namespaces that are allowed in traffic sent through a room (see the Allowable Traffic section of this document).

16.5 Field Standardization

Field Standardization for Data Forms (XEP-0068) [48] defines a process for standardizing the fields used within Data Forms qualified by a particular FORM_TYPE. Within MUC, there are four uses of such forms: room registration (the "muc#register" FORM_TYPE), requesting voice and approving voice requests ("muc#request"), room configuration ("muc#roomconfig"), and service discovery extensions for room information ("muc#roominfo"). The reserved fields are defined below.

16.5.1 muc#register FORM_TYPE

```
<form_type>
  <name>http://jabber.org/protocol/muc#register</name>
  <doc>XEP-0045</doc>
  <desc>
    Forms enabling user registration with a
    Multi-User Chat (MUC) room or admin approval
    of user registration requests.
  </desc>
  <field
     var='muc#register_allow'
     type='boolean'
     label='Allow this person to register with the room?'/>
  <field
      var='muc#register_email'
      type='text-single'
      label='Email Address'/>
  <field
      var='muc#register_faqentry'
      type='text-multi'
      label='FAQ Entry'/>
  <field
      var='muc#register_first'
```

[48] XEP-0068: Field Data Standardization for Data Forms <https://xmpp.org/extensions/xep-0068.html>.

```
      type='text-single'
      label='Given Name'/>
  <field
      var='muc#register_last'
      type='text-single'
      label='Family Name'/>
  <field
      var='muc#register_roomnick'
      type='text-single'
      label='Desired Nickname'/>
  <field
      var='muc#register_url'
      type='text-single'
      label='A Web Page'/>
</form_type>
```

16.5.2 muc#request FORM_TYPE

```
<form_type>
  <name>http://jabber.org/protocol/muc#request</name>
  <doc>XEP-0045</doc>
  <desc>
    Forms enabling voice requests in a
    Multi-User Chat (MUC) room or admin
    approval of such requests.
  </desc>
  <field var='muc#role'
         type='list-single'
         label='Requested role'/>
  <field var='muc#jid'
         type='jid-single'
         label='User ID'/>
  <field var='muc#roomnick'
         type='text-single'
         label='Room Nickname'/>
  <field var='muc#request_allow'
         type='boolean'
         label='Whether to grant voice'/>
</form_type>
```

16.5.3 muc#roomconfig FORM_TYPE

```
<form_type>
  <name>http://jabber.org/protocol/muc#roomconfig</name>
  <doc>XEP-0045</doc>
  <desc>
    Forms enabling creation and configuration of
```

```
    a Multi-User Chat (MUC) room.
  </desc>
  <field
      var='muc#maxhistoryfetch'
      type='text-single'
      label='Maximum Number of History Messages Returned by Room'/>
  <field
      var='muc#roomconfig_allowpm'
      type='list-single'
      label='Roles that May Send Private Messages'/>
  <field
      var='muc#roomconfig_allowinvites'
      type='boolean'
      label='Whether to Allow Occupants to Invite Others'/>
  <field
      var='muc#roomconfig_changesubject'
      type='boolean'
      label='Whether to Allow Occupants to Change Subject'/>
  <field
      var='muc#roomconfig_enablelogging'
      type='boolean'
      label='Whether to Enable Public Logging of Room Conversations'/>
  <field
      var='muc#roomconfig_getmemberlist'
      type='list-multi'
      label='Roles and Affiliations that May Retrieve Member List'/>
  <field
      var='muc#roomconfig_lang'
      type='text-single'
      label='Natural Language for Room Discussions'/>
  <field
      var='muc#roomconfig_pubsub'
      type='text-single'
      label='XMPP URI of Associated Publish-Subscribe Node'/>
  <field
      var='muc#roomconfig_maxusers'
      type='list-single'
      label='Maximum Number of Room Occupants'/>
  <field
      var='muc#roomconfig_membersonly'
      type='boolean'
      label='Whether to Make Room Members-Only'/>
  <field
      var='muc#roomconfig_moderatedroom'
      type='boolean'
      label='Whether to Make Room Moderated'/>
  <field
      var='muc#roomconfig_passwordprotectedroom'
      type='boolean'
```

```
      label='Whether␣a␣Password␣is␣Required␣to␣Enter'/>
  <field
      var='muc#roomconfig_persistentroom'
      type='boolean'
      label='Whether␣to␣Make␣Room␣Persistent'/>
  <field
      var='muc#roomconfig_presencebroadcast'
      type='list-multi'
      label='Roles␣for␣which␣Presence␣is␣Broadcasted'/>
  <field
      var='muc#roomconfig_publicroom'
      type='boolean'
      label='Whether␣to␣Allow␣Public␣Searching␣for␣Room'/>
  <field
      var='muc#roomconfig_roomadmins'
      type='jid-multi'
      label='Full␣List␣of␣Room␣Admins'/>
  <field
      var='muc#roomconfig_roomdesc'
      type='text-single'
      label='Short␣Description␣of␣Room'/>
  <field
      var='muc#roomconfig_roomname'
      type='text-single'
      label='Natural-Language␣Room␣Name'/>
  <field
      var='muc#roomconfig_roomowners'
      type='jid-multi'
      label='Full␣List␣of␣Room␣Owners'/>
  <field
      var='muc#roomconfig_roomsecret'
      type='text-single'
      label='The␣Room␣Password'/>
  <field
      var='muc#roomconfig_whois'
      type='list-single'
      label='Affiliations␣that␣May␣Discover␣Real␣JIDs␣of␣Occupants'/>
</form_type>
```

16.5.4 muc#roominfo FORM_TYPE

```
<form_type>
  <name>http://jabber.org/protocol/muc#roominfo</name>
  <doc>XEP-0045</doc>
  <desc>
    Forms enabling the communication of extended service discovery
    information about a Multi-User Chat (MUC) room.
  </desc>
```

```
  <field
      var='muc#maxhistoryfetch'
      type='text-single'
      label='Maximum␣Number␣of␣History␣Messages␣Returned␣by␣Room'/>
  <field
      var='muc#roominfo_contactjid'
      type='jid-multi'
      label='Contact␣Addresses␣(normally,␣room␣owner␣or␣owners)'/>
  <field
      var='muc#roominfo_description'
      type='text-single'
      label='Short␣Description␣of␣Room'/>
  <field
      var='muc#roominfo_lang'
      type='text-single'
      label='Natural␣Language␣for␣Room␣Discussions'/>
  <field
      var='muc#roominfo_ldapgroup'
      type='text-single'
      label='An␣associated␣LDAP␣group␣that␣defines
␣␣␣␣␣␣␣␣␣␣␣␣␣␣room␣membership;␣this␣should␣be␣an␣LDAP
␣␣␣␣␣␣␣␣␣␣␣␣␣␣Distinguished␣Name␣according␣to␣an
␣␣␣␣␣␣␣␣␣␣␣␣␣␣implementation-specific␣or
␣␣␣␣␣␣␣␣␣␣␣␣␣␣deployment-specific␣definition␣of␣a
␣␣␣␣␣␣␣␣␣␣␣␣␣␣group.'/>
  <field
      var='muc#roominfo_logs'
      type='text-single'
      label='URL␣for␣Archived␣Discussion␣Logs'/>
  <field
      var='muc#roominfo_occupants'
      type='text-single'
      label='Current␣Number␣of␣Occupants␣in␣Room'/>
  <field
      var='muc#roominfo_subject'
      type='text-single'
      label='Current␣Discussion␣Topic'/>
  <field
      var='muc#roominfo_subjectmod'
      type='boolean'
      label='The␣room␣subject␣can␣be␣modified␣by␣participants'/>
</form_type>
```

16.6 Status Codes Registry

16.6.1 Process

The XMPP Registrar maintains a registry at <https://xmpp.org/registrar/mucstatus.html> that defines values for the 'code' attribute of the <status/> element when qualified by the

'http://jabber.org/protocol/muc#user' namespace.
In order to submit new values to this registry, the registrant shall define an XML fragment of the following form and either include it in the relevant XMPP Extension Protocol or send it to the email address <registrar@xmpp.org>:

```
<statuscode>
  <number>
    the three-digit code number
  </number>
  <stanza>
    the stanza type of which it is a child (message or presence)
  </stanza>
  <context>
    the use case or situation in which the status is used
  </context>
  <purpose>
    a natural-language description of the meaning
  </purpose>
  <child>
    the descriptive child element (reserved for future use)
  </child>
</statuscode>
```

The registrant may register more than one status code at a time, each contained in a separate <statuscode/> element.

16.6.2 Initial Submission

As part of this document, the following status codes are registered:

```
<statuscode>
  <number>100</number>
  <stanza>message or presence</stanza>
  <context>Entering a room</context>
  <purpose>
    Inform user that any occupant is allowed to see the user's full
       JID
  </purpose>
</statuscode>
<statuscode>
  <number>101</number>
  <stanza>message (out of band)</stanza>
  <context>Affiliation change</context>
  <purpose>
    Inform user that his or her affiliation changed while not in the
    room
```

```
  </purpose>
</statuscode>
<statuscode>
  <number>102</number>
  <stanza>message</stanza>
  <context>Configuration change</context>
  <purpose>
    Inform occupants that room now shows unavailable members
  </purpose>
</statuscode>
<statuscode>
  <number>103</number>
  <stanza>message</stanza>
  <context>Configuration change</context>
  <purpose>
    Inform occupants that room now does not show unavailable members
  </purpose>
</statuscode>
<statuscode>
  <number>104</number>
  <stanza>message</stanza>
  <context>Configuration change</context>
  <purpose>
    Inform occupants that a non-privacy-related room configuration
    change has occurred
  </purpose>
</statuscode>
<statuscode>
  <number>110</number>
  <stanza>presence</stanza>
  <context>Any room presence</context>
  <purpose>
    Inform user that presence refers to itself
  </purpose>
</statuscode>
<statuscode>
  <number>170</number>
  <stanza>message or initial presence</stanza>
  <context>Configuration change</context>
  <purpose>
    Inform occupants that room logging is now enabled
  </purpose>
</statuscode>
<statuscode>
  <number>171</number>
  <stanza>message</stanza>
  <context>Configuration change</context>
  <purpose>
    Inform occupants that room logging is now disabled
```

```
  </purpose>
</statuscode>
<statuscode>
  <number>172</number>
  <stanza>message</stanza>
  <context>Configuration change</context>
  <purpose>
    Inform occupants that the room is now non-anonymous
  </purpose>
</statuscode>
<statuscode>
  <number>173</number>
  <stanza>message</stanza>
  <context>Configuration change</context>
  <purpose>
    Inform occupants that the room is now semi-anonymous
  </purpose>
</statuscode>
<statuscode>
  <number>201</number>
  <stanza>presence</stanza>
  <context>Entering a room</context>
  <purpose>
    Inform user that a new room has been created
  </purpose>
</statuscode>
<statuscode>
  <number>210</number>
  <stanza>presence</stanza>
  <context>Entering a room, changing nickname, etc.</context>
  <purpose>
    Inform user that service has assigned or modified occupant's
    roomnick
  </purpose>
</statuscode>
<statuscode>
  <number>301</number>
  <stanza>presence</stanza>
  <context>Removal from room</context>
  <purpose>
    A user has been banned from the room
  </purpose>
</statuscode>
<statuscode>
  <number>303</number>
  <stanza>presence</stanza>
  <context>Exiting a room</context>
  <purpose>
    Inform all occupants of new room nickname
```

```
  </purpose>
</statuscode>
<statuscode>
  <number>307</number>
  <stanza>presence</stanza>
  <context>Removal from room</context>
  <purpose>
    A user has been kicked from the room
  </purpose>
</statuscode>
<statuscode>
  <number>321</number>
  <stanza>presence</stanza>
  <context>Removal from room</context>
  <purpose>
    A user is being removed from the room
    because of an affiliation change
  </purpose>
</statuscode>
<statuscode>
  <number>322</number>
  <stanza>presence</stanza>
  <context>Removal from room</context>
  <purpose>
    A user is being removed from the room
    because the room has been changed to members-only and the
    user is not a member
  </purpose>
</statuscode>
<statuscode>
  <number>332</number>
  <stanza>presence</stanza>
  <context>Removal from room</context>
  <purpose>
    A user is being removed from the room
    because the MUC service is being shut down
  </purpose>
</statuscode>
<statuscode>
  <number>333</number>
  <stanza>presence</stanza>
  <context>Removal from room</context>
  <purpose>
    A user was removed because of an error reply (for example
    when an s2s link fails between the MUC and the removed users
        server).
  </purpose>
</statuscode>
```

16.7 URI Query Types

As authorized by XMPP URI Query Components (XEP-0147) [49], the XMPP Registrar maintains a registry of queries and key-value pairs for use in XMPP URIs (see <https://xmpp.org/registrar/querytypes.html>).

16.7.1 join

The "join" querytype is registered as a MUC-related action, with an optional key of "password".

Listing 208: Join Action: IRI/URI

```
xmpp:coven@chat.shakespeare.lit?join
```

The application MUST either present an interface enabling the user to provide a room nickname or populate the room nickname based on configured preferences or nickname discovery.

Listing 209: Join Action: Resulting Stanza

```
<presence to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>
```

The join action MAY include a password for the room. Naturally, access to a URI that includes a room password MUST be appropriately controlled.

Listing 210: Join Action with Password: IRI/URI

```
xmpp:coven@chat.shakespeare.lit?join;password=cauldronburn
```

Listing 211: Join Action with Password: Resulting Stanza

```
<presence to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'>
    <password>cauldronburn</password>
  </x>
</presence>
```

The following submission registers the "join" querytype.

```
<querytype>
  <name>join</name>
  <proto>http://jabber.org/protocol/muc</proto>
  <desc>enables joining a multi-user chat room</desc>
  <doc>XEP-0045</doc>
```

[49] XEP-0147: XMPP URI Query Components <https://xmpp.org/extensions/xep-0147.html>.

```
  <keys>
    <key>
      <name>password</name>
      <desc>the password required to enter the room</desc>
    </key>
  </keys>
</querytype>
```

16.7.2 invite

The "invite" querytype is registered as a MUC-related action, with an optional key of "jid".

Listing 212: Invite Action: IRI/URI

```
xmpp:coven@chat.shakespeare.lit?invite;jid=hecate@shakespeare.lit
```

If the joining user is not yet in the room, the application MUST send two stanzas: the first to join the room and the second to invite the other individual. If the joining user is in the room already, the application shall send only the invitation stanza.

Listing 213: Invite Action: Resulting Stanza(s)

```
<presence to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>

<message to='coven@chat.shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite to='hecate@shakespeare.lit'/>
  </x>
</message>
```

The URI can include multiple invitees:

Listing 214: Invite Action With Multiple Invitees: IRI/URI

```
xmpp:coven@chat.shakespeare.lit?invite;jid=hecate@shakespeare.lit;jid=
   bard@shakespeare.lit
```

Listing 215: Invite Action With Multiple Invitees: Resulting Stanza

```
<message to='coven@chat.shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite to='hecate@shakespeare.lit'/>
    <invite to='bard@shakespeare.lit'/>
  </x>
</message>
```

The URI can also include a password:

Listing 216: Invite Action With Password: IRI/URI

```
xmpp:coven@chat.shakespeare.lit?invite;jid=hecate@shakespeare.lit;
   password=cauldronburn
```

Listing 217: Invite Action With Password: Resulting Stanza(s)

```
<presence to='coven@chat.shakespeare.lit/thirdwitch'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>

<message to='coven@chat.shakespeare.lit'>
  <x xmlns='http://jabber.org/protocol/muc#user'>
    <invite to='hecate@shakespeare.lit'/>
    <password>cauldronburn</password>
  </x>
</message>
```

The following submission registers the ”invite” querytype.

```
<querytype>
  <name>invite</name>
  <proto>http://jabber.org/protocol/muc</proto>
  <desc>enables simultaneously joining a groupchat room and inviting
     others</desc>
  <doc>XEP-0045</doc>
  <keys>
    <key>
      <name>jid</name>
      <desc>the Jabber ID of the invitee</desc>
    </key>
    <key>
      <name>password</name>
      <desc>the password required to enter a multi-user chat room</
         desc>
    </key>
  </keys>
</querytype>
```

17 Business Rules

17.1 Addresses

In order to provide consistency regarding the addresses captured in room JIDs and occupant JIDs, Room IDs MUST match the Nodeprep profile of Stringprep and Room Nicknames MUST

match the Resourceprep profile of Stringprep (both of these are defined in RFC 6122 [50]). As explicitly stated in RFC 6122, both the Room ID (node) and Room Nickname (resource) portions of an Occupant JID MUST be of non-zero length. In addition, a MUC service MUST NOT allow empty or invisible Room Nicknames (i.e., Room Nicknames that consist only of one or more space characters).
It is up to the service implementation whether it will further restrict roomnicks (e.g., by applying case folding routines, the Nodeprep profile of stringprep, or other restrictions).

17.2 Message

1. If an occupant wants to send a message to all other occupants, a MUC client MUST set the 'type' attribute to a value of "groupchat". A service MAY ignore messages that are improperly typed, or reject them with a <bad-request/> error.

2. If a MUC service receives a message directed to the room or to a single occupant from a user who has a role of "none", the service MUST NOT deliver the message and SHOULD return the message to the sender with a <forbidden/> error.

3. If a MUC service receives a message directed to a room that does not exist or is not yet unlocked, the service SHOULD return the message to the sender with an <item-not-found/> error.

4. A MUC service SHOULD pass extended information (e.g., an XHTML version of the message body) through to occupants unchanged; however, a MUC service MAY disallow message specific extensions (see the Allowable Traffic section of this document).

5. A MUC client MAY generate extensions that conform to Chat State Notifications (XEP-0085) [51] specification; however, a MUC service MAY disallow these extensions (see the Allowable Traffic section of this document).

17.3 Presence

1. The presence stanza used to join a room MUST NOT possess a 'type' attribute, i.e., it must be available presence. A MUC service MUST NOT treat a presence stanza with a 'type' attribute (e.g., a presence probe) as a request to join the room.

[50] RFC 6122: Extensible Messaging and Presence Protocol (XMPP): Address Format <http://tools.ietf.org/html/rfc6122>.
[51] XEP-0085: Chat State Notifications <https://xmpp.org/extensions/xep-0085.html>.

2. The presence stanza used to exit a room MUST possess a 'type' attribute whose value is "unavailable". A MUC service MUST NOT treat a presence stanza with a 'type' attribute whose value is other than "unavailable" (e.g., a presence probe) as a request to exit the room.

3. If a MUC service receives a Basic MUC Protocol join request from a client that is already joined, it MUST treat it as a full synchronization request, and send to the client everything that would be sent to it on a normal join. The MUC service SHOULD send a presence update to the other participants if the join presence is different from the client's previous presence.

4. A MUC service MAY handle presence probes sent to the room JID <room@service> or an occupant JID <room@service/nick> (e.g, these might be sent by an occupant's home server to determine if the room is still online or to synchronize presence information if the user or the user's server has gone offline temporarily or has started sharing presence again, as for instance when Stanza Interception and Filtering Technology (XEP-0273) [52] is used).

5. A room MUST silently ignore unavailable presence received from a user who has a role of "none".

6. Only the MUC service itself SHOULD generate data about roles, affiliations, full JIDs, or status codes qualified by the 'http://jabber.org/protocol/muc#user' namespace (based on information the service knows about occupants, e.g., roles, or as a result of actions taken by a moderator or room administrator). A client SHOULD NOT presume to generate such information. If a MUC service receives such extended presence information from an occupant, it MUST NOT reflect it to other occupants. (A client MAY generate extended presence information qualified by the 'http://jabber.org/protocol/muc#user' namespace in order to supply a password, but naturally this is not reflected to other occupants.)

7. A MUC service SHOULD allow all other presence information to pass through, although it MAY choose to block extended presence information; see the Allowable Traffic section of this document.

8. In order to inform occupants of room roles and affiliations, and to make it easier for clients to track the current state of all users in the room, MUC service implementations MUST provide role and affiliation data (and, if allowed by the room configuration, full

[52] XEP-0273: Stanza Interception and Filtering Technology <https://xmpp.org/extensions/xep-0273.html>.

JID) in all presence stanzas, including presence stanzas of type ”unavailable” sent when a user exits the room for any reason.

9. If a role or affiliation is revoked, the service MUST note that fact by sending an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child element with the 'role' and/or 'affiliation' attributes set to a value that indicates the loss of the relevant status. All future presence stanzas for the occupant MUST include the updated role and affiliation, until and unless they change again.

10. If the service includes an occupant's JabberID in the MUC presence extension, the value of the 'jid' attribute MUST be the full JID (not the bare JID).

11. A client MAY send a custom exit message if desired (as is often done in IRC channels) by including a <status/> element in the presence stanza of type ”unavailable” sent when exiting a room.

17.4 IQ

1. MUC is designed for sharing of messages and presence, not IQs. An IQ sent to the JID of the room itself is handled by the room and is not reflected to all of the room occupants.

2. If an occupant wants to send an IQ stanza to another user in a non-anonymous room, the sender SHOULD send the request directly to the recipient's bare JID or full JID, rather than attempting to send the request through the room (i.e., via the recipient's occupant JID).

3. If an occupant wants to send an IQ stanza to another user in a semi-anonymous room, the sender can direct the stanza to the recipient's occupant JID and the service SHOULD forward the stanza to the recipient's real JID. However, the MUC service MUST NOT reveal the sender's real JID to the recipient at any time, nor reveal the recipient's real JID to the sender.

4. A MUC client MUST send only the 'affiliation' attribute or the 'role' attribute in the <item/> element contained within an IQ set qualified by the 'http://jabber.org/protocol/muc#admin' namespace; if a moderator, admin, or owner attempts to modify both the affiliation and role of the same item in the same IQ set, the service MUST return a <bad-request/> error to the sender. However, the MUC service MAY modify a role based on a change to an affiliation and thus MAY send

presence updates that include both a modified role and a modified affiliation.

5. In IQ sets regarding roles, a MUC client MUST include the 'nick' attribute only; in IQ results regarding roles, a MUC service MUST include the 'nick', 'role', 'affiliation', and 'jid' attributes (with the value of the latter set to the user's full JID).

6. In IQ sets regarding affiliations, a MUC client MUST include the 'jid' attribute only (with the value set to the bare JID); in IQ results regarding affiliations, a MUC service MUST NOT include the 'role' attribute, MUST include the 'affiliation' attribute and the 'jid' attribute (with the value set to the bare JID), and SHOULD include the 'nick' attribute (except if the affiliation is "outcast", since outcasts SHOULD NOT have reserved nicknames).

18 Implementation Notes

The following guidelines are intended to assist client and service developers in creating MUC implementations.

18.1 Services

1. In handling messages sent by visitors in a moderated room, a MUC service MAY queue each message for approval by a moderator and MAY inform the sender that the message is being held for approval; however, such behavior is OPTIONAL, and definition of a message approval protocol (e.g., using Data Forms as defined in Data Forms (XEP-0004) [53]) is out of scope for this document.

2. Out of courtesy, a MUC service MAY send an out-of-room <message/> if a user's affiliation changes while the user is not in the room; the message SHOULD be sent from the room to the user's bare JID, MAY contain a <body/> element describing the affiliation change, and MUST contain a status code of 101.

3. There is no requirement that a MUC service shall provide special treatment for users of the older groupchat 1.0 protocol, such as messages that contain equivalents to the extended presence information that is qualified by the 'http://jabber.org/protocol/muc#user' namespace.

4. Room types MAY be configured in any combination. A MUC service MAY support or allow any desired room types or combinations thereof. There is no guarantee that any

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

such combination is sensible.

5. A MUC service MAY limit the number of configuration options presented to an owner after initial configuration has been completed, e.g. because certain options cannot take effect without restarting the service.

6. A MUC service MAY provide an interface to room creation and configuration (e.g., in the form of a special XMPP entity or a Web page), so that the ostensible room owner is actually the application instead of a human user.

7. A MUC service MAY choose to make available a special in-room resource that provides an interface to administrative functionality (e.g., a "user" named "ChatBot"), which occupants could interact with directly, thus enabling admins to type '/command parameter' in a private message to that "user". Obviously this kind of implementation would require the service to add a 'ChatBot' user to the room when it is created, and to prevent any occupant from having the nickname 'ChatBot' in the room. This might be difficult to ensure in some implementations or deployments. In any case, any such interface is OPTIONAL.

8. A MUC service MAY choose to discard extended presence information that is attached to a <presence/> stanza before reflecting the presence change to the occupants of a room. That is, an implementation MAY choose to reflect only the <show/>, <status/>, and <priority/> child elements of the presence element as specified in the XML schema for the 'jabber:client' namespace, with the result that presence "changes" in extended namespaces (e.g., gabber:x:music:info) are not passed through to occupants. If a service prohibits certain extended namespaces, it SHOULD provide a description of allowable traffic at the well-known Service Discovery node 'http://jabber.org/protocol/muc#traffic' as described in the Allowable Traffic section of this document.

9. A MUC service MAY choose to discard extended information attached to a <message/> stanza before reflecting the message to the occupants of a room. An example of such extended information is the lightweight text markup specified by XHTML-IM (XEP-0071) [54]. If a service prohibits certain extended namespaces, it SHOULD provide a description of allowable traffic at the well-known Service Discovery node 'http://jabber.org/protocol/muc#traffic' as described in the Allowable Traffic section of this document.

[54] XEP-0071: XHTML-IM <https://xmpp.org/extensions/xep-0071.html>.

10. A MUC service MAY choose to "lock down" room nicknames (e.g., hardcoding the room nickname to the bare JID of the occupant). If so, the service MUST treat the locked nickname as a reserved room nickname and MUST support the protocol specified in the Discovering Reserved Room Nickname section of this document.

18.1.1 Allowable Traffic

As noted, a service (more precisely, a properly-configured room) MAY discard some or all extended namespaces attached to <message/> and <presence/> stanzas that are intended for reflection from the sender through the room to all of the room occupants. If the room does so, it SHOULD enable senders to discover the list of allowable extensions by sending a disco#info query to the well-known Service Discovery node 'http://jabber.org/protocol/muc#traffic', returning one <feature/> element for each namespace supported in the result. If the room does not allow any extended namespaces, it MUST return an empty query as specified in Service Discovery (XEP-0030) [55]. If the room does not support the "#traffic" node, it MUST return a <feature-not-implemented/> error in response to queries sent to the 'http://jabber.org/protocol/muc#traffic' node.
The following example shows a room that allows the 'http://jabber.org/protocol/xhtml-im' and 'http://jabber.org/protocol/rosterx' namespaces only, but no other extended namespaces.

Listing 218: User Queries Service Regarding Allowable Namespaces

```
<iq from='wiccarocks@shakespeare.lit/laptop'
    to='heath@chat.shakespeare.lit'
    id='allow1'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/muc#traffic'/>
</iq>
```

Listing 219: Service Returns Allowable Namespaces

```
<iq from='heath@chat.shakespeare.lit'
    to='wiccarocks@shakespeare.lit/laptop'
    id='allow1'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/muc#traffic'>
    <feature var='http://jabber.org/protocol/xhtml-im'/>
    <feature var='http://jabber.org/protocol/rosterx'/>
  </query>
</iq>
```

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

If a service does not discard any namespaces, it MUST return a <service-unavailable/> error:

Listing 220: Service Returns Service Unavailable

```
<iq from='heath@chat.shakespeare.lit'
    to='wiccarocks@shakespeare.lit/laptop'
    id='allow1'
    type='error'>
  <error type='cancel'>
    <service-unavailable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

18.1.2 Ghost Users

Deployment experience has shown that sometimes a user can appear to be an occupant in a room even though the user's real JID has gone offline since joining. Such users are called "ghosts". To help prevent ghost users, a MUC service SHOULD remove a user if the service receives a delivery-related error in relation to a stanza it has previously sent to the user (in this context, the delivery-related errors are <gone/>, <item-not-found/>, <recipient-unavailable/>, <redirect/>, <remote-server-not-found/>, and <remote-server-timeout/>). A MUC service MAY also use XMPP Ping (XEP-0199) [56] or similar methods to periodically check the availability of room occupants. If the MUC service determines that the user has gone offline, it must treat the user as if the user had itself sent unavailable presence.

18.2 Clients

1. Clients MAY present room roles by showing ad-hoc groups for each role within a room roster. This will enable occupants to clearly visualize which occupants are moderators, participants, and visitors. However, such a representation is OPTIONAL.

2. Clients MAY implement a variety of interface styles that provide "shortcuts" to functionality such as changing one's nickname, kicking or banning users, discovering an occupant's full JID, or changing the subject. One option consists of IRC-style commands such as '/nick', '/kick', '/ban', and '/whois'; another is to enable a user to right-click items in a room roster. All such interface styles are OPTIONAL. However, for convenience, a mapping of IRC commands to MUC protocols is provided below.

[56] XEP-0199: XMPP Ping <https://xmpp.org/extensions/xep-0199.html>.

18.2.1 IRC Command Mapping

Internet Relay Chat clients use a number of common ”shortcut” commands that begin with a forward slash, such as ’/nick’ and ’/ban’. The following table provides a mapping of IRC-style commands to MUC protocols, for use by clients that wish to support such functionality.

Command	Function	MUC protocol
/ban <roomnick> [comment]	bans user with that roomnick from this room (client translates roomnick to bare JID)	<iq id=’someid’ to=’room@service’ type=’set’> <query xmlns=’http://jabber.org/protocol/muc#admin’> <item affiliation=’outcast’ jid=’bare-jid-of-user’> <reason>comment</reason> </item> </query> </iq>
/invite <jid> [comment]	invites user with that JID to this room	<message to=’room@service’> <x xmlns=’http://jabber.org/protocol/muc#user’> <invite to=’jid’> <reason>comment</reason> </invite> </x> </message>
/join <roomname> [pass]	joins room on this service (roomnick is same as nick in this room)	<presence to=’room@service/nick’> <x xmlns=’http://jabber.org/protocol/muc#user’> <password>pass</password> </x> </presence>
/kick <roomnick> [comment]	kicks user with that roomnick from this room	<iq id=’someid’ to=’room@service’ type=’set’> <query xmlns=’http://jabber.org/protocol/muc#admin’> <item nick=’roomnick’ role=’none’> <reason>comment</reason> </item> </query> </iq>
/msg <roomnick> <foo>	sends private message ”foo” to roomnick	<message to=’room@service/nick’ type=’chat’> <body>foo</body> </message>
/nick <newnick>	changes nick in this room to ”newnick”	<presence to=’room@service/newnick’/>

Command	Function	MUC protocol
/part [comment]	exits this room (some IRC clients also support /leave)	<presence to='room@service/nick' type='unavailable'> <status>comment</status> </presence>
/topic <foo>	changes subject of this room to "foo"	<message to='room@service' type='groupchat'> <subject>foo</subject> </message>

Note: Because MUC roomnicks follow the Resourceprep profile of stringprep, they are allowed to contain a space character, whereas IRC nicknames do not. Although a given client MAY support quotation characters for this purpose (resulting in commands such as '/ban "king lear" insanity is no defense'), most common quotation characters (such as " and ') are also allowed by Resourceprep, thus leading to added complexity and potential problems with quotation of roomnicks that contain both spaces and quotation characters. Therefore it is NOT RECOMMENDED for XMPP clients to support IRC-style shortcut commands with roomnicks that contain space characters.
Note: Many XMPP clients also implement a '/me ' command as described in The /me Command (XEP-0245) [57]. This command does not result in any MUC or IRC protocol action and is therefore not shown in the foregoing table.

18.2.2 Presence Subscriptions

In XMPP, presence subscriptions are used to share network availability information between end users (see RFC 6121 [58]). In XMPP groupchat, the presence employed is of a special type: directed presence rather than presence subscriptions. However, an IM user can also subscribe to the presence of a chatroom, which introduces the useful property that the user will be notified when a room comes back online after a crash or other outage, thus facilitating the feature of automatically re-joining the room. Although MUC clients and servers have not traditionally implemented presence subscriptions, developers are encouraged to consider adding such support.

[57] XEP-0245: The /me Command <https://xmpp.org/extensions/xep-0245.html>.
[58] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

19 XML Schemas

19.1 http://jabber.org/protocol/muc

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://jabber.org/protocol/muc'
    xmlns='http://jabber.org/protocol/muc'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0045: http://www.xmpp.org/extensions/xep-0045.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='x'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='history' minOccurs='0'/>
        <xs:element name='password' type='xs:string' minOccurs='0'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='history'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='maxchars' type='xs:int' use='optional'/>
          <xs:attribute name='maxstanzas' type='xs:int' use='optional'
              />
          <xs:attribute name='seconds' type='xs:int' use='optional'/>
          <xs:attribute name='since' type='xs:dateTime' use='optional'
              />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>
```

```
</xs:schema>
```

19.2 http://jabber.org/protocol/muc#user

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://jabber.org/protocol/muc#user'
    xmlns='http://jabber.org/protocol/muc#user'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0045: http://www.xmpp.org/extensions/xep-0045.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='x'>
    <xs:complexType>
      <xs:choice minOccurs='0' maxOccurs='unbounded'>
        <xs:element ref='decline' minOccurs='0'/>
        <xs:element ref='destroy' minOccurs='0'/>
        <xs:element ref='invite' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element ref='item' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='password' type='xs:string' minOccurs='0'/>
        <xs:element ref='status' minOccurs='0' maxOccurs='unbounded'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name='decline'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='reason' minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='from' type='xs:string' use='optional'/>
      <xs:attribute name='to' type='xs:string' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='destroy'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='reason' minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='jid' type='xs:string' use='optional'/>
```

```
    </xs:complexType>
  </xs:element>

  <xs:element name='invite'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='reason' minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='from' type='xs:string' use='optional'/>
      <xs:attribute name='to' type='xs:string' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='item'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='actor' minOccurs='0'/>
        <xs:element ref='continue' minOccurs='0'/>
        <xs:element ref='reason' minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='affiliation' use='optional'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='admin'/>
            <xs:enumeration value='member'/>
            <xs:enumeration value='none'/>
            <xs:enumeration value='outcast'/>
            <xs:enumeration value='owner'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name='jid' type='xs:string' use='optional'/>
      <xs:attribute name='nick' type='xs:string' use='optional'/>
      <xs:attribute name='role' use='optional'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='moderator'/>
            <xs:enumeration value='none'/>
            <xs:enumeration value='participant'/>
            <xs:enumeration value='visitor'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

  <xs:element name='actor'>
    <xs:complexType>
      <xs:simpleContent>
```

```
        <xs:extension base='empty'>
          <xs:attribute name='jid' type='xs:string' use='optional'/>
          <xs:attribute name='nick' type='xs:string' use='optional'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='continue'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='thread' type='xs:string' use='optional'/
              >
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='status'>
    <xs:complexType>
      <xs:attribute name='code' use='required'>
        <xs:simpleType>
          <xs:restriction base='xs:int'>
            <xs:minInclusive value='100'/>
            <xs:maxInclusive value='999'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

  <xs:element name='reason' type='xs:string'/>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

19.3 http://jabber.org/protocol/muc#admin

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
```

```
    targetNamespace='http://jabber.org/protocol/muc#admin'
    xmlns='http://jabber.org/protocol/muc#admin'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0045: http://www.xmpp.org/extensions/xep-0045.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='query'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='item' maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='item'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='actor' minOccurs='0'/>
        <xs:element ref='reason' minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='affiliation' use='optional'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='admin'/>
            <xs:enumeration value='member'/>
            <xs:enumeration value='none'/>
            <xs:enumeration value='outcast'/>
            <xs:enumeration value='owner'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name='jid' type='xs:string' use='optional'/>
      <xs:attribute name='nick' type='xs:string' use='optional'/>
      <xs:attribute name='role' use='optional'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='moderator'/>
            <xs:enumeration value='none'/>
            <xs:enumeration value='participant'/>
            <xs:enumeration value='visitor'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
```

```
  </xs:element>

  <xs:element name='actor'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='jid' type='xs:string' use='optional'/>
          <xs:attribute name='nick' type='xs:string' use='optional'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='reason' type='xs:string'/>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

19.4 http://jabber.org/protocol/muc#owner

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://jabber.org/protocol/muc#owner'
    xmlns='http://jabber.org/protocol/muc#owner'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0045: http://www.xmpp.org/extensions/xep-0045.html
    </xs:documentation>
  </xs:annotation>

  <xs:import
      namespace='jabber:x:data'
      schemaLocation='http://www.xmpp.org/schemas/x-data.xsd'/>

  <xs:element name='query'>
    <xs:complexType>
      <xs:choice xmlns:xdata='jabber:x:data' minOccurs='0'>
        <xs:element ref='xdata:x'/>
```

```
        <xs:element ref='destroy'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name='destroy'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='password' type='xs:string' minOccurs='0'/>
        <xs:element name='reason' type='xs:string' minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='jid' type='xs:string' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

20 Acknowledgements

The author would like to especially recognize the following individuals for their many helpful comments on various drafts of this proposal: Gaston Dombiak, Joe Hildebrand, Craig Kaes, Jacek Konieczny, Peter Millard, Jean-Louis Seguineau, Alexey Shchepin, David Sutton, and David Waite. Thanks also to members of the XSF Technical Review Team for their edits and suggestions, in particular Peter Mount and Luca Tagliaferri. In addition, more people than the author can count have have provided feedback in the jdev@conference.jabber.org chat room and on the standards@xmpp.org and muc@xmpp.org mailing lists.