DGA Game XMPP Network Protocol Specification

Zhanat S. Skokbayev

D6a

DISTANT GROUND ATTACK

This section is currently a draft, and is subject to change.

Version: 0.1
Date: 10.01.2026
Description: First publication.

Contents at a Glance

This section is currently a draft, and is subject to change.

Table of Contents

List of Tables

List of Examples

Document History

This section is currently a draft, and is subject to change.

Version	Date	Description	Author
0.0.1	24.03.2019	First draft.	Zhanat S. Skokbayev
0.0.2	01.09.2019	First presentation.	Zhanat S. Skokbayev
0.1	10.01.2026	First publication.	Zhanat S. Skokbayev

Preface

This section is currently a draft, and is subject to change.

DGA Game XMPP Network Protocol Specification ('the Specification') defines matters and guidelines for the DGA Game XMPP Network Protocol ('the Network Protocol'), which is an XMPP protocol extension for a game called 'DGA Distant Ground Attack' ('DGA', 'the DGA Game', or simply 'the Game').

This chapter introduces common topics related to the Specification.

Audience

The Specification is intended to be read by programmers, artists, testers, producers, managers, players, and everybody else involved in design, development, testing, and use of the Game. However, software developers, programmers, and architects are the primary audience of the Specification.

Legal

Prerequisites

XMPP is a protocol for streaming XML elements over a network in order to exchange structured yet extensible data in near-real-time. We assume that you are familiar with the very basics of computer networking, common Internet applications (such as email and the World Wide Web), and structured data formats (such as HTML). The first place to start here is The XMPP Standards Foundation’s website.

The DGA Game is based on the Java Platform and written in the Java™ Programming Language. DGA’s software architecture is closely related to the Java Platform’s technologies. If you are new to Java, spend some time getting up to speed on the language and platform; a good place to start is dev.java/learn.

Also, each topic in this Specification provides some background information, but in general, we assume you have a basic knowledge of the technologies each Java Platform’s feature works with. Another field important for understanding the Specification is game development. We assume that you have some basic understanding in these fields.

Related Documentation

For more related information, see the following documents of the Game:

- *DGA Game Concepts Guide* specifies design concepts and guidelines of the DGA Game.
- *DGA Game Architecture Guide* provides all necessary information about DGA's systems, software, and network architectures.
- *DGA Game Design Guide* details *DGA Game Concepts Guide* and serves as a blueprint from which the Game is being built.

Internationalisation and Localisation

From the beginning, DGA is implemented as an internationalised computer software addressed to the global audience and international markets. The following locale is default for the DGA Game and its projects:

Locale	Name	Description
en-EU	English in the European Union (European English, EU English).	The English language as it is accepted in the European Union as the shared standard usage of Ireland and the United Kingdom. As a general rule, Irish/British English is preferred, and Americanisms that are liable not to be understood by speakers of Irish/British English should be avoided. However, bearing in mind that a considerable proportion of the target readership may be made up of non-native speakers, very colloquial Irish/British usage should also be avoided. Although, the International System of Units (SI System) is used by default. (EU Language Rules; EU Guidelines for Translating into English)

At the same time, DGA is intended to be localised for as many languages and countries as possible. DGA strives to communicate with every player in his/her native language or in a language of his/her preference. Nevertheless, this communication has to be well implemented.

Since the Network Protocol is based on XSF's standards written in American English, the Specification follows the style of the original language whenever appropriate.

Terminological Conventions

Throughout this document, we use the following terminological conventions:

Massive Vs. Massively Multiplayer Online Game

Massive versus *Massively* Multiplayer Online Game (MMOG) is a highly debatable topic. Laying aside the aspect of grammar, we prefer expanding *MMOG* as *Massive Multiplayer Online Game* because it represents a more general notion. The word *massive* refers to a multiplayer online game, which can be not only massively multiplayer (i.e. massive in quantity of players acting within the same online game world), but also huge in functionality, gameplay, network capabilities, virtual economy, and other parameters of the game. Whereas, the form *massively multiplayer* indicates only that there are numerous players in the game, i.e. this form has a more narrow meaning. Moreover, a massive multiplayer game can be either massively multiplayer or not. The modern game encyclopedias allow both forms, although they use *massively multiplayer* in the narrow, specific sense. (ECGG, p. 596)

Typographic Conventions

Throughout this document, we use the following typographic conventions:

Convention	Meaning	Example
Boldface	Boldface type indicates a new term defined in text below or in the glossary. Also it marks graphical user interface elements associated with an action.	**Heaven** signifies night and day, cold and heat, times and seasons. From the **File** menu, choose **Open Project**.
Italic	Italic type indicates book titles, emphasis, terms in the text or placeholder variables for which you supply particular values.	Read Chapter 6 in the *User's Guide*. Do *not* save the file. The command to remove a file is `rm` *`filename`*.
`Monospace`	Monospace type indicates the names of files and directories, commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.	Edit your `.login` file. Use `ls -a` to list all files. *`machine_name`*`% you have mail.`
very important	Underlined type indicates texts of especial importance.	**The Commander** stands for the virtues of wisdom, sincerity, benevolence, courage and strictness.
(Book, page)	Letter code and number in round brackets indicates the index of a text resource through the Bibliography and the page number(s) within the text.	(ECGG, p. 319) (XTDG, pp. 27-29)

Basic Standards

This section is currently a draft, and is subject to change.

The Network Protocol is based on the XMPP Protocol (Extensible Messaging and Presence Protocol), which is maintained by the XMPP Standards Foundation (XSF) and Internet Engineering Task Force (IETF).

XMPP Standards

The Network Protocol is defined by the following basic XSF and IETF’s standards (Request for Comments, RFC):

1. RFC 6120 'Extensible Messaging and Presence Protocol (XMPP): Core', which specifies the XMPP Protocol as an application profile of the Extensible Markup Language (XML) that enables the near-real-time exchange of structured yet extensible data between any two or more network entities. This document defines XMPP’s core protocol methods: setup and teardown of XML streams, channel encryption, authentication, error handling, and communication primitives for messaging, network availability ('presence'), and request-response interactions. This document obsoletes RFC 3920.
2. RFC 6121 'Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence', which specifies extensions to core features of the XMPP Protocol that provide basic instant messaging (IM) and presence functionality in conformance with the requirements in RFC 2779. This document obsoletes RFC 3921.
3. RFC 7395 'An Extensible Messaging and Presence Protocol (XMPP) Subprotocol for WebSocket', which specifies a binding for the XMPP Protocol over a WebSocket transport layer. A WebSocket binding for XMPP provides higher performance than the current HTTP binding for XMPP.
4. RFC 7590 'Use of Transport Layer Security (TLS) in the Extensible Messaging and Presence Protocol (XMPP)', which specifies the use of Transport Layer Security (TLS) in the XMPP Protocol. This document updates RFC 6120.
5. RFC 7622 'Extensible Messaging and Presence Protocol (XMPP): Address Format', which specifies the address format for the XMPP Protocol, including support for code points outside the ASCII range. This document partially obsoletes RFC 6122.

XMPP Guidelines

The Specification is arranged according to the following XSF’s standard guidelines (XMPP Extension Protocols, XEP):

1. XEP-0001 'XMPP Extension Protocols', which defines the standards process followed by the XMPP Standards Foundation.
2. XEP-0134 'XMPP Design Guidelines', which defines best practices for the intelligent design of Jabber/XMPP protocols and other XMPP extensions.
3. XEP-0143 'Guidelines for Authors of XMPP Extension Protocols', which provides information intended to assist authors of XMPP Extension Protocols.
4.

XEP-0045 'Multi-User Chat', which specifies 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.

5. XEP-0196 'User Gaming', which specifies an XMPP protocol extension for communicating information about the games a user plays.
6. XEP-xxxx 'Multi-User Gaming', which specifies an XMPP protocol extension for multi-user gaming.

XMPP Guidelines Notices

The Specification gives credit to the notices of the following XMPP Extension Protocols the DGA Game XMPP Network Protocol is based on.

XEP-0045: Multi-User Chat

“

Abstract	***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.***
Authors	*Peter Saint-Andre*
Copyright	*© 2002 – 2024 XMPP Standards Foundation.*
Status	*Stable*
	NOTICE: The protocol defined herein is a Stable Standard of the XMPP Standards Foundation. Implementations are encouraged and the protocol is appropriate for deployment in production systems, but some changes to the protocol are possible before it becomes a Final Standard.
Type	*Standards Track*
Version	*1.35.1 (2024-09-17)*

XEP-0196: User Gaming

“

Abstract	***This document defines an XMPP protocol extension for communicating information about the games a user plays.***
Authors	*Peter Saint-Andre*
Copyright	*© 2006 – 2008 XMPP Standards Foundation.*
Status	*Deferred*
	WARNING: This document has been automatically Deferred after 12 months of inactivity in its previous Experimental state. Implementation of the protocol described herein is not recommended for production systems. However, exploratory implementations are encouraged to resume the standards process.
Type	*Standards Track*
Version	*0.3 (2008-09-25)*

XEP-xxxx: Multi-User Gaming

Abstract	***This document defines an XMPP protocol extension for multi-user gaming.***
Authors	*Torsten Grote, Arne König, Günther Nieß*
Copyright	*© 2008 – 2009 XMPP Standards Foundation.*
Status	*ProtoXEP*
	WARNING: This document has not yet been accepted for consideration or approved in any official manner by the XMPP Standards Foundation, and this document is not yet an XMPP Extension Protocol (XEP). If this document is accepted as a XEP by the XMPP Council, it will be published at https://xmpp.org/extensions/ and announced on the <standards@xmpp.org> mailing list.
Type	*Standards Track*
Version	*0.0.3 (2009-04-20)*

XEP-xxxx

DGA MMOG

This section is currently a draft, and is subject to change.

Abstract	***This document defines an XMPP protocol extension for the DGA MMORPG Game Network Protocol.***
Authors	*Zhanat S. Skokbayev*
Copyright	*© 2026 Zhanat S. Skokbayev at The FLEISS Software Foundation. SEE LEGAL NOTICES.*
Status	*ProtoXEP*
	WARNING: This document has not yet been accepted for consideration or approved in any official manner by the XMPP Standards Foundation, and this document is not yet an XMPP Extension Protocol (XEP). If this document is accepted as a XEP by the XMPP Council, it will be published at https://xmpp.org/extensions/ and announced on the <standards@xmpp.org> mailing list.
Type	*Standards Track*
Version	*0.1 (2026-01-10)*

1. Introduction

Next generation of communication networks provides sufficient capabilities to implement XMPP for massive multiplayer online games. A massive multiplayer online game (MMOG) refers to video games that allow a large number of players, typically from hundreds to thousands, to participate simultaneously over Internet connections on the same game server. These games usually feature complex gameplay, network capabilities, virtual economy, and other functionalities on a large scale. Still, XMPP mostly lacks this kind of support. Therefore, this document (Massive Multiplayer Online Gaming or MMOG) describes an extension protocol for MMO game playing over XMPP.

The Network Protocol is designed as self-sufficient to play this kind of games. Its basic mechanics are completely based on the specification of XEP-0045 'Multi-User Chat' (MUC) intended for many-to-many chat messaging. 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 is not compatible to the groupchat 1.0 protocol, but provides advanced features such as invitations, room moderation and administration, and specialized room types.

The MMOG protocol also integrates the specifications of XEP-xxxx 'Multi-User Gaming' (MUG) and XEP-0196 'User Gaming' extension protocols. The MUG protocol has been conceived as not sufficient to play games; it just describes a basic protocol framework, which game-specific protocols can use. The User Gaming protocol specifies an extended presence payload format that communicates information about the games a user plays.

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 and its successors: RFC 2810, RFC 2811, RFC 2812, RFC 2813).

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).
- Moderation of individual messages.
- Encryption of messages sent through a room.
- Advanced features such as attaching files to a room, integrating whiteboards, and using MMOG rooms as a way to manage the signalling for multi-user audio or video conferencing (see Multiparty Jingle in XEP-0272).
- Interaction between MMOG deployments and foreign chat systems (e.g., gateways to IRC or to legacy IM systems).
- Mirroring or replication of rooms among multiple MMOG 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/mmog' 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", whereas "none" represents no affiliation; affiliation is distinct from role. An affiliation lasts across a user’s visits to a room.

Ammunition

Ammunition (informally ammo) is the material fired, scattered, dropped or detonated from any weapon.

Angular velocity

Angular velocity refers to how fast an object rotates or revolves relative to another point, i.e. how fast the angular position or orientation of an object changes with time. There are two types of angular velocity: orbital angular velocity and spin angular velocity. Spin angular velocity refers to how fast a rigid body rotates with respect to its centre of rotation. Orbital angular velocity refers to how fast a point object revolves about a fixed origin, i.e. the time rate of change of its angular position relative to the origin. Spin angular velocity is independent of the choice of origin, in contrast to orbital angular velocity which depends on the choice of origin.
Angular velocity is measured in angle per unit time, e.g. radians per second (angle replacing distance from linear velocity with time in common). The SI unit of angular velocity is expressed as radians per second with the radian having a dimensionless value of unity, thus the SI units of angular velocity are listed as $1/s$ or $s{-1}$. Angular velocity is usually represented by the symbol ω (omega), sometimes Ω. By convention, positive angular velocity indicates counter-clockwise rotation, while negative is clockwise.

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.

Battle Royale Match (BRM)

A type of matches while a match is starting in a room when there are at least two players waiting for it, but the match does not finish until the last player stays in the match for some time and other players were destroyed. If another player enters the match at this moment then the battle continues. Players can battle in the match each against other personally or within their proper teams. According to this property there are two subtypes of battle royale matches:

1. Last-Man-Standing Match (LMSM)
2. Last-Team-Standing Match (LTSM).

Character

A character (sometimes known as a fictional character) is a person or other being in a narrative (such as a novel, play, television series, film, or video game). In video games characters are represented in two essential forms: player characters (PCs) and non-player characters (NPCs).

Defeat

An act of being defeated by the opponent players or teams in a match. Being defeated means that opponent players physically destroyed (damaged) this player, these players, their team, or the opponent players managed to achieve some set of victorious requirements.

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.

Game

In general sense, a game is a structured form of play, contest, competition, usually undertaken for entertainment or fun, and sometimes used as an educational tool. In a more specific sense, a game is a XEP which defines the rules of a match.

Gamefield

An area, place, location where a game match is being conducted. If the game's rules suppose an open conflict between players then "gamefield" is the synonym for "battlefield".

Groupchat (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 room or match.

Initiator

The entity that started a game.

Invitation

A special message sent from one user to another asking the recipient to join a room; the invitation can be sent directly (see XEP-0249) 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".

Linear Velocity

Velocity is the rate of change in the position of an object in a specific range of time. When the object moves along the straight path, the velocity associated with it is termed as linear velocity. It is given as the ratio of distance covered to time:

$v = x/t$, where,

v - linear velocity,

x - distance covered,

t - time taken to cover the distance x.

Logging

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

Match

Represents a specific instance of a game played in a room.

Match Type

A category of matches having a common set of characteristics.

Member

A user who is on the "whitelist" 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.

MUG

The multi-user gaming protocol for a game playing over XMPP.

MMOG

The multi-user gaming protocol for a massive multi-player online game, which is played over XMPP, especially designed to be played simultaneously by a large number of players and has complex gameplay, network capabilities, virtual economy, etc.

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 *richardIII@york.lit* is allowed to log in simultaneously as the nick *"King"* in the *characters@thewarsoftheroses.shakespeare.lit* game room from both *richardIII@york.lit/berwick* and *richardIII@york.lit/bosworth*). Multi-session nicks are not currently defined in this document.

Node

A node defines an internal and integral part of a character. Nodes represent different parts the character consists of, and they are used to describe the character's position, state, etc.

Non-Player Character (NPC)

A non-player character (NPC) is any character in a game which is not controlled by a player. In video games this usually means a character controlled by the computer (instead of the player) that has a predetermined set of behaviours that potentially will impact game play, but not necessarily be true artificial intelligence.

Occupant

Any user who is in a room (this is an "abstract class" and does not correspond to any specific role). An occupant can become a player or spectator within a match.

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.

Owner

A privileged entity that owns a game.

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".

Player

A user in a match who has a defined game team.

Player Character (PC)

A player character (also known as PC and playable character) is a fictional character in a video game whose actions are directly controlled by a player of the game rather than the rules of the game. The characters that are not controlled by a player are called non-player characters (NPCs).

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", "none" for 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 play matches 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 resource part 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 matches and occupants in a room.

Server

An XMPP server that may or may not have associated with it a gaming service.

Service

A host that offers gaming capabilities; often but not necessarily a sub-domain of an XMPP server (e.g., *mmog.jabber.org*).

Single Battle Match (SBM)

A type of matches while a match is starting in a room only when a list of players according to some set of requirements has been completed. The battle match is finishing when the player(s) of a team destroyed the last player(s) of the opposing team(s) or when the team achieved some set of victorious requirements (the team wined / gained a victory and defeated other teams).

Spectator

An occupant who does not actually plays games but watches them. Spectators do not have voice (in contrast to players).

Subject

A temporary gaming topic within a room.

Team

A side which a player belongs to within a match. For example, it can be "black" and "white" as in chess, "Montecchi" and "Capuleti", "proponent" and "opponent", and so on.

The team "all" is a reserved name for querying the list of active players and MUST NOT be redefined by games for other purposes.

Victory

An act of defeating the opponent players or teams in a match. Defeating means that opponent players were physically destroyed (damaged) or some set of victorious requirements has been achieved.

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.

Weapon

A weapon, arm or armament is any implement or device that can be used with intent to inflict damage or harm.

4.2. Room Types

Fully-Anonymous Room

A room in which the full JIDs or bare JIDs of occupants cannot be discovered by anyone, including the room owner; contrast with Non-Anonymous Room and Semi-Anonymous Room.

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 and Fully-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 Fully-Anonymous Room and 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. Room Status

There are different room statuses as follows:

Created

A room before the initial room configuration is done.

Active

A room that is currently in use.

Adjourned

A "saved" room, i.e. rooms archived for future notices.

4.4. Match Status

There are different match statuses as follows:

Created

A match before the initial match configuration is done.

Inactive

The status before and after a match, turns are not possible, options can be changed.

Active

The status within a match, turns are possible, options cannot be changed.

Paused

The status halted within a match, turns are not possible, options cannot be changed.

Adjourned

A "saved" match, i.e. matches archived for future notices.

4.5. Dramatis Personae

Most of the examples in this Specification use the scenario of King Richard III and Henry Tudor, Earl of Richmond fought in Act V of Shakespeare's Richard III, taking place at the Battle of Bosworth Field and represented here as the *"bosworth@games.shakespeare.lit"* room. The characters are as follows:

Table 1: Dramatis Personae

Room Nickname	Full JID	Role	Team
king	*richardiii@shakespeare.lit/desktop*	Participant	York
sirwilliam	*wcatesby@shakespeare.lit/laptop*	Participant	York
earl1	*harritudur@shakespeare.lit/pda*	Participant	Lancaster
earlofderby	*thomasstanley@shakespeare.lit/cell*	Participant	Lancaster
queenconsort	*elizabeth@shakespeare.lit/tablet*	Participant	None
hastings	*whastings@shakespeare.lit/smartphone*	Participant	None
buckingham	*buckingham@shakespeare.lit/notebook*	Participant	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").

MMOG takes the second approach.

MMOG also defines two different associations: long-lived affiliations and session-specific roles. These two association types are distinct from each other in MMOG, 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:

Table 2: Roles

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).

Table 3: Privileges Associated With Roles

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).

Table 4: Initial Role Based on Affiliation

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, an MMOG 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.

Table 5: Role State Chart

>	**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
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 affiliation is not hardcoded and is left up to the implementation.

5.2. Matches

Owners are allowed to do what they like (saving/loading, change match options, etc.) except in unmoderated matches. This match type restricts the use of owner privileges to specific room statuses. Users with no affiliation SHALL NOT enter members-only matches. Besides that, these users have the same privileges as members.

Table 6: Owner Privileges Overview

Room Type	Configure	Save/Load	Grant Member	Revoke Member	Assign Role	Revoke Role
moderated match	inactive	all status	all status	all status	all status	all status
unmoderated match	inactive	inactive	all status	inactive	inactive and paused	inactive and paused

5.3. 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 "whitelist" 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).

5.3.1. 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.

Table 7: Privileges Associated With Affiliations

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).

5.3.2. 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, an MMOG 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.

Table 8: Affiliation State Chart

>	**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

An MMOG implementation MUST support Service Discovery (XEP-0030) ("disco"), Service Discovery Extensions (XEP-0128) and Jabber Search (XEP-0055). Any entity can complete the following disco-related use cases.

6.1. Discovering an MMOG Service

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

Example 1. Entity Queries Server for Associated Services

```
<iq from='harritudur@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.

Example 2. Server Returns Disco Items Result

```
<iq from='shakespeare.lit'
    id='h7ns81g'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='games.shakespeare.lit'
          name='Massively Multiplayer Online Gaming Service'/>
  </query>
</iq>
```

6.2. Discovering the Features Supported by an MMOG Service

An entity may wish to discover if a service implements the Massively Multiplayer Online Gaming protocol; in order to do so, it sends a service discovery information ("disco#info") query to the MMOG service’s JID.

Example 3. Entity Queries Gaming Service for MMOG Support via Disco

```
<iq from='harritudur@shakespeare.lit/pda'
    id='lx09df27'
    to='games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

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

Example 4. Service Returns Disco Info Result

```
<iq from='games.shakespeare.lit'
    id='lx09df27'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='game'
        name='Massively Multiplayer Online Gaming Service'
        type='multi-user'/>
    <feature var='jabber:iq:search'/>
    <feature var='http://jabber.org/protocol/mmog'/>
    <feature var='http://jabber.org/protocol/mmog/dga'/>
  </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 game service would consist of the game rooms hosted by the service.

Example 5. Entity Queries Chat Service for Rooms

```
<iq from='harritudur@shakespeare.lit/pda'
    id='zb8q41f4'
    to='games.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).

Example 6. Service Returns Disco Items Result

```
<iq from='games.shakespeare.lit'
    id='zb8q41f4'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='england@games.shakespeare.lit'
          name='england'/>
    <item jid='ireland@games.shakespeare.lit'
          name='ireland'/>
    <item jid='scotland@games.shakespeare.lit'
          name='scotland'/>
    <item jid='wales@games.shakespeare.lit'
          name='wales'/>
  </query>
</iq>
```

If the full list of rooms is large (see XEP-0030 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)) to indicate that the list is not the full result set.

Example 7. Service Returns Limited List of Disco Items Result

```
<iq from='games.shakespeare.lit'
    id='hx51v49s'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='buckingham@games.shakespeare.lit'/>
    <item jid='cambridgeshire@games.shakespeare.lit'/>
    <item jid='clarence@games.shakespeare.lit'/>
    <item jid='derby@games.shakespeare.lit'/>
    <item jid='dorset@games.shakespeare.lit'/>
    <item jid='gloucester@games.shakespeare.lit'/>
    <item jid='norfolk@games.shakespeare.lit'/>
    <item jid='oxford@games.shakespeare.lit'/>
    <item jid='surrey@games.shakespeare.lit'/>
    <item jid='york@games.shakespeare.lit'/>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first index='0'>buckingham@games.shakespeare.lit</first>
      <last>york@games.shakespeare.lit</last>
      <count>37</count>
    </set>
  </query>
</iq>
```

6.4. Search for Rooms

It is RECOMMENDED that a user uses Jabber Search (XEP-0055) to search for active or adjourned rooms. At first the user needs to discover what search fields are supported by the service:

Example 8. Client Requests Search Fields from Service

```
<iq type='get'
    from='thomasstanley@shakespeare.lit/cell'
    to='games.shakespeare.lit'
    id='search1'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'/>
</iq>
```

The service MUST then return the possible search fields to the user, and MAY include instructions:

Example 9. Service Returns Search Form to Client

```
<iq type='result'
    from='games.shakespeare.lit'
    to='thomasstanley@shakespeare.lit/cell'
    id='search1'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'>
    <instructions>
      Use the enclosed form to search. If your Jabber client does not
      support Data Forms, visit http://web.games.shakespeare.lit/
    </instructions>
    <x xmlns='jabber:x:data' type='form'>
      <title>Room Search</title>
      <instructions>
        Please provide the following information
        to search for active or adjourned matches.
      </instructions>
      <field type='hidden'
             var='FORM_TYPE'>
        <value>jabber:iq:search</value>
      </field>
      <field type='text-single'
             label='Room Name'
             var='mmog#roomsearch_name'/>
      <field type='boolean'
             label='Saved Rooms'
             var='mmog#roomsearch_saved'/>
      <field type='list-single'
             label='Free Game Roles'
             var='mmog#roomsearch_roles'>
        <option label='1'><value>1</value></option>
        <option label='2'><value>2</value></option>
        <option label='3'><value>3</value></option>
        <option label='4'><value>4</value></option>
        <option label='5'><value>5</value></option>
      </field>
      <field type='list-single'
             label='Maximum Number of Occupants'
             var='mmog#roomsearch_max_occupants'>
        <option label='1'><value>2</value></option>
        <option label='2'><value>3</value></option>
        <option label='3'><value>4</value></option>
        <option label='4'><value>5</value></option>
        <option label='5'><value>10</value></option>
        <option label='5'><value>20</value></option>
      </field>
      <field type='list-single'
             label='Game Category'
             var='mmog#roomsearch_category'>
        <option label='Role Playing Games'><value>board</value></option>
        <option label='Bulletin Board Games'><value>cards</value></option>
      </field>
      <field type='list-multi'
             label='Games'
             var='mmog#roomsearch_game'>
        <option label='DGA'><value>http://jabber.org/protocol/mmog#dga</value></option>
        <option label='DDT'><value>http://jabber.org/protocol/mmog#ddt</value></option>
      </field>
    </x>
```

The Saved Room option allows to search for active or adjourned rooms (see Room Status). The Game Category field is to classify the game and to be able to only search for certain types of games. Every game protocol MUST define its category in the corresponding game XEP.

After having received the possible search fields, the user MAY then submit a search request, specifying values for any desired fields:

Example 10. User Submits Search Form

```
<iq type='set'
    from='thomasstanley@shakespeare.lit/cell'
    to='games.shakespeare.lit'
    id='search2'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:search</value>
      </field>
      <field var='status'>
        <value>active</value>
      </field>
      <field var='game'>
        <value>http://jabber.org/protocol/mmog/dga</value>
      </field>
    </x>
  </query>
</iq>
```

The submitting entity MAY submit the 'category' or 'game' field but MUST NOT submit both. Sending an empty form adds up to searching for all games.

The service SHOULD then return a list of search results that match the values provided:

Example 11. Service Returns Search Results

```
<iq type='result'
    from='games.shakespeare.lit'
    to='thomasstanley@shakespeare.lit/cell'
    id='search2'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'>
    <x xmlns='jabber:x:data' type='result'>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:search</value>
      </field>
      <reported>
        <field var='status' label='Match Status' type='list-single'/>
        <field var='category' label='Game Category' type='list-single'/>
        <field var='game' label='Game NS' type='text-single'/>
        <field var='jid' label='Jabber ID' type='jid-single'/>
      </reported>
      <item>
        <field var='status'><value>active</value></field>
        <field var='category'><value>rpg</value></field>
        <field var='game'><value>http://jabber.org/protocol/mmog/dga</value></field>
        <field var='jid'><value>england@games.shakespeare.lit</value></field>
      </item>
      ...
    </x>
  </query>
</iq>
```

If the full list of rooms is large, the service MAY return only a partial list of rooms. If it does so, it SHOULD include a <set/> element (as defined in Result Set Management (XEP-0059)) to indicate that the list is not the full result set.

6.5. Querying for Room Information

Using the disco#info protocol, an entity may also query a specific game 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 Considerations for details).

Example 12. Entity Queries for Information about a Specific Game Room

```
<iq from='harritudur@shakespeare.lit/pda'
    id='ik3vs715'
    to='england@games.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:

Example 13. Room Returns Disco Info Result

```
<iq from='england@games.shakespeare.lit'
    id='ik3vs715'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='game'
        name='england'
        type='multi-user'/>
    <feature var='http://jabber.org/protocol/mmog'/>
    <feature var='http://jabber.org/protocol/mmog/dga'/>
    <feature var='http://jabber.org/protocol/mmog#stable_id'/>
    <feature var='mmog_passwordprotected'/>
    <feature var='mmog_hidden'/>
    <feature var='mmog_temporary'/>
    <feature var='mmog_open'/>
    <feature var='mmog_unmoderated'/>
    <feature var='mmog_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 game room MAY return more detailed information in its disco#info response using Service Discovery Extensions (XEP-0128), identified by inclusion of a hidden FORM_TYPE field whose value is "http://jabber.org/protocol/mmog#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:

Example 14. Room Returns Extended Disco Info Result

```
<iq from='england@games.shakespeare.lit'
    id='ik3vs715'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='game'
        name='england'
        type='multi-user'/>
    <feature var='http://jabber.org/protocol/mmog'/>
    <feature var='http://jabber.org/protocol/mmog/dga'/>
    <feature var='mmog_passwordprotected'/>
    <feature var='mmog_hidden'/>
    <feature var='mmog_temporary'/>
    <feature var='mmog_open'/>
    <feature var='mmog_unmoderated'/>
    <feature var='mmog_nonanonymous'/>
    <x xmlns='jabber:x:data' type='result'>
      <field var='FORM_TYPE' type='hidden'>
        <value>http://jabber.org/protocol/mmog#roominfo</value>
      </field>
      <field var='mmog#roominfo_description'
             label='Description'>
        <value>The House of York!</value>
      </field>
      <field var='mmog#roominfo_changesubject'
             label='Occupants May Change the Subject'>
        <value>true</value>
      </field>
      <field var='mmog#roominfo_contactjid'
             label='Contact Addresses'>
        <value>richardiii@shakespeare.lit</value>
      </field>
      <field var='mmog#roominfo_subject'
             label='Current Battle Topic'>
        <value>Fights</value>
      </field>
      <field var='mmog#roomconfig_changesubject'
             label='Subject can be modified'>
        <value>true</value>
      </field>
      <field var='mmog#roominfo_occupants'
             label='Number of occupants'>
        <value>3</value>
      </field>
      <field var='mmog#roominfo_ldapgroup'
             label='Associated LDAP Group'>
        <value>cn=roses,dc=shakespeare,dc=lit</value>
      </field>
      <field var='mmog#roominfo_lang'
             label='Language of discussion'>
        <value>en</value>
      </field>
      <field var='mmog#roominfo_logs'
             label='URL for discussion logs'>
        <value>http://www.shakespeare.lit/gamelogs/england/</value>
      </field>
      <field var='mmog#maxhistoryfetch'
             label='Maximum Number of History Messages Returned by Room'>
```

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 mmog#roomconfig FORM_TYPE can be included in the extended service discovery fields (as shown above for the "mmog#roomconfig_changesubject" field).

> Note: The foregoing extended service discovery fields for the 'http://jabber.org/protocol/mmog#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.

6.6. Querying for Room Items

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

Example 15. Entity Queries for Items Associated with a Specific Game Room

```
<iq from='harritudur@shakespeare.lit/pda'
    id='kl2fax27'
    to='england@games.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.

Example 16. Room Returns Disco Items Result (Items are Public)

```
<iq from='england@games.shakespeare.lit'
    id='kl2fax27'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='england@games.shakespeare.lit/richardiii'/>
    <item jid='england@games.shakespeare.lit/wcatesby'/>
  </query>
</iq>
```

> Note: These <item/> elements are qualified by the disco#items namespace, not the MMOG 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 XEP-0030.

Example 17. Room Returns Empty Disco Items Result (Items are Private)

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

6.7. Querying a Room Occupant

If a non-occupant attempts to send a disco request to an address of the form <room@service/nick>, an MMOG 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.

6.8. Discovering Client Support for MMOG

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

Example 18. Entity Queries Contact Regarding MMOG Support

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

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

Example 19. Contact Returns Disco Info Result

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

Example 20. Entity Queries Contact for Current Rooms

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

Example 21. Contact Returns Room Query Result

```
<iq from='wcatesby@shakespeare.lit/laptop'
    id='gp7w61v3'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://jabber.org/protocol/mmog#rooms'>
    <item jid='england@games.shakespeare.lit'/>
    <item jid='characters@thewarsoftheroses.shakespeare.lit'/>
  </query>
</iq>
```

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

```
    ...
    <item jid='england@games.shakespeare.lit'
          name='sirwilliam'/>
    ...
```

If this information is private, the user MUST return an empty <query/> element, in accordance with XEP-0030.

6.9. Announcing a Running Game

The client MAY implement User Gaming (XEP-0196) to announce running games. To publish a running game the user sends:

Example 22. User Publishes Gaming Information

```
<iq type='set' from='richardiii@shakespeare.lit/desktop' id='publish1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:gaming:0'>
      <item id='1feea9cceec2537e1b561e66d45bc566e276f22f'>
        <game xmlns='urn:xmpp:gaming:0'>
          <name>The Wars of the Roses</name>
          <char_name>twr</char_name>
          <server_name>Massively Multiplayer Online Gaming Service</server_name>
          <server_address>games.shakespeare.lit</server_address>
          <uri>xmpp:dga@games.shakespeare.lit?
play;game=http://jabber.org/protocol/mmog/dga</uri>
        </game>
      </item>
    </publish>
  </pubsub>
</iq>
```

When the user stops playing the game (i.e. leaves the game room), the user's client SHOULD send an empty <game/> element with the same ItemID:

Example 23. User Publishes Gaming Information On Exit

```
<iq type='set' from='richardiii@shakespeare.lit/desktop' id='publish2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:gaming:0'>
      <item id='1feea9cceec2537e1b561e66d45bc566e276f22f'>
        <game xmlns='urn:xmpp:gaming:0'/>
      </item>
    </publish>
  </pubsub>
</iq>
```

7. Occupant Use Cases

The main actor in a Massively Multiplayer Online Gaming environment is the occupant, who can be said to be located "in" a multi-user game 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 fulfil 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 and therefore MMOG).
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/mmog#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) 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 MMOG service MUST send it events in the following order:

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 MMOG Protocol

In order to participate in the discussions held in a multi-user game room, a user MUST first become an occupant by entering the room.

MMOG clients MUST signal their ability to speak the MMOG protocol by including in the initial presence stanza an empty <game/> element qualified by the 'http://jabber.org/protocol/mmog' namespace (note the absence of the '#user' fragment):

Example 24. User Seeks to Enter a Room (Massively Multiplayer Online Gaming)

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='n13mt3l'
    to='england@games.shakespeare.lit/earl1'>
  <game
    xmlns='http://jabber.org/protocol/mmog'
    var='http://jabber.org/protocol/mmog/dga'/>
</presence>
```

In this example, a user with a full JID of "*harritudur@shakespeare.lit/pda*" has requested to enter the room "england" on the "games.shakespeare.lit" gaming service with a room nickname of "earl1".

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:

Example 25. No Nickname Specified

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

Before attempting to enter the room, an MMOG-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 an MMOG service receives a <game/> 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 MMOG. 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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#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. *(NB)*

Example 26. Service Sends Match State and Presence from Existing Occupants to New Occupant

```
<presence
    from='england@games.shakespeare.lit'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <status>active</status>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      <x xmlns='jabber:x:data' type='submit'>
        ...
      </x>
    </state>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/king'
    id='3DCB0401-D7CF-4E31-BE05-EDF8D057BFBD'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='owner' role='moderator' team='york'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/sirwilliam'
    id='C2CD9EE3-8421-431E-854A-A2AD0CE2E23D'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='admin' role='moderator' team='york'/>
  </game>
</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 "king" (who is a room owner) and a user with a room nickname of "sirwilliam" (who is a room admin).

Unless the room is configured to not broadcast presence from new occupants below a certain affiliation level (as controlled by the "mmog#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).

Example 27. Service Sends New Occupant’s Presence to All Occupants

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='27C55F89-1C6A-459A-9EB5-77690145D624'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member' role='participant'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/earl1'
    id='9E757BAE-8AC8-4093-AA9C-407F6AEF15D6'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member' role='participant'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/earl1'
    id='026B3509-2CCE-4D69-96D6-25F41FFDC408'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member' role='participant'/>
    <status code='110'/>
  </game>
</presence>
```

In this example, initial room presence is being sent from the new occupant (earl1) 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).

In particular, if roomnicks are locked down then the service MUST do one of the following.

If the user has connected using a "groupchat 1.0" client (as indicated on joining the room by the lack of the MMOG extension), then the service SHOULD deny the nickname change request and return a presence stanza of type "error" with a <not-acceptable/> error condition:

Example 28. Service Denies Room Join Because Roomnicks Are Locked Down

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='ng91xs69'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.shakespeare.lit' type='cancel'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

If the user has connected using an MMOG client (as indicated on joining the room by inclusion of the MMOG extension), then 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.

Example 29. Service Sends New Occupant's Presence to New Occupant

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member' role='participant'/>
    <status code='110'/>
    <status code='210'/>
  </game>
</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 MAY 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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with a 'jid' attribute specifying the occupant's full JID:

Example 30. Service Sends Full JID to All Occupants

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='17232D15-134F-43C8-9A29-61C20A64B236'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none'
        jid='harritudur@shakespeare.lit/pda'
        role='participant'/>
  </game>
</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:

Example 31. Service Sends New Occupant's Presence to New Occupant

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member' role='participant'/>
    <status code='100'/>
    <status code='110'/>
    <status code='210'/>
  </game>
</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 "mmog#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:

Example 32. Service Denies Access Because No Password Provided

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog' 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).

Example 33. User Provides Password On Entering a Room

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='djn4714'
    to='england@games.shakespeare.lit/earl1'>
  <game
      xmlns='http://jabber.org/protocol/mmog'
      var='http://jabber.org/protocol/mmog/dga'>
    <password>dieuetmondroit</password>
  </game>
</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:

Example 34. Service Denies Access Because User Is Not on Member List

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.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:

Example 35. Service Denies Access Because User is Banned

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.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:

Example 36. Service Denies Access Because of Nick Conflict

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.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:

Example 37. Service Informs User that Room Occupant Limit Has Been Reached

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.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:

Example 38. Service Denies Access Because Room Does Not (Yet) Exist

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.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:

Example 39. Service Sends New Occupant’s Presence to New Occupant

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='n13mt3l'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member' role='participant'/>
    <status code='100'/>
    <status code='110'/>
    <status code='170'/>
    <status code='210'/>
  </game>
</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 game service implementation or specific deployment depending on local service policy or room configuration.

Example 40. Delivery of Discussion History

```
<message
    from='england@games.shakespeare.lit/king'
    id='162BEBB1-F6DB-4D9A-9BD8-CFDCC801A0B2'
    to='norfolk@shakespeare.lit/broom'
    type='groupchat'>
  <body>A horse! a horse! my kingdom for a horse!</body>
  <delay xmlns='urn:xmpp:delay'
     from='england@games.shakespeare.lit'
     stamp='2019-09-07T18:22:02Z'/>
</message>

<message
    from='england@games.shakespeare.lit/sirwilliam'
    id='90057840-30FD-4141-AA44-103EEDF218FC'
    to='norfolk@shakespeare.lit/broom'
    type='groupchat'>
  <body>Withdraw, my lord; I'll help you to a horse.</body>
  <delay xmlns='urn:xmpp:delay'
     from='england@games.shakespeare.lit'
     stamp='2019-09-07T18:22:10Z'/>
</message>

<message
    from='england@games.shakespeare.lit/earl1'
    id='77E07BB0-55CF-4BD4-890E-3F7C0E686BBD'
    to='norfolk@shakespeare.lit/broom'
    type='groupchat'>
  <body>I think there be six Richmonds in the field.</body>
  <delay xmlns='urn:xmpp:delay'
     from='england@games.shakespeare.lit'
     stamp='2019-09-07T18:22:23Z'/>
</message>
```

Discussion history messages MUST be stamped with Delayed Delivery (XEP-0203) 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 XEP-0203 supersedes the older 'jabber:x:delay' namespace defined in Legacy Delayed Delivery (XEP-0091); 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) element that notes the original full JID of the sender by means of the "ofrom" address type:

Example 41. Discussion History Message with Original From

```
<message
    from='england@games.shakespeare.lit/king'
    id='162BEBB1-F6DB-4D9A-9BD8-CFDCC801A0B2'
    to='norfolk@shakespeare.lit/broom'
    type='groupchat'>
  <body>Five have I slain today instead of him.</body>
  <delay xmlns='urn:xmpp:delay'
     from='england@games.shakespeare.lit'
     stamp='2019-09-07T18:22:32Z'/>
  <addresses xmlns='http://jabber.org/protocol/address'>
    <address type='ofrom' jid='richardiii@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:

Table 9: History Management Attributes

Attribute	Datatype	Meaning
maxchars	int	Limit the total number of characters in the history to "game" (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 "game".
seconds	int	Send only the messages received in the last "game" 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)).

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 MMOG 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.

Example 42. User Requests Limit on Number of Characters in History

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='n13mt3l'
    to='england@games.shakespeare.lit/earl1'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <history maxchars='65000'/>
  </game>
</presence>
```

Example 43. User Requests Limit on Number of Messages in History

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='n13mt3l'
    to='england@games.shakespeare.lit/earl1'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <history maxstanzas='20'/>
  </game>
</presence>
```

Example 44. User Requests History in Last 3 Minutes

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='n13mt3l'
    to='england@games.shakespeare.lit/earl1'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <history seconds='180'/>
  </game>
</presence>
```

Example 45. User Requests All History Since the Beginning of the Unix Era

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='n13mt3l'
    to='england@games.shakespeare.lit/earl1'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <history since='1970-01-01T00:00:00Z'/>
  </game>
</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.

Example 46. User Requests No History

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='n13mt3l'
    to='england@games.shakespeare.lit/earl1'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <history maxchars='0'/>
  </game>
</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.

Example 47. Service Informs New Occupant of Room Subject

```
<message
    from='england@games.shakespeare.lit/sirwilliam'
    id='F437C672-D438-4BD3-9BFF-091050D32EE2'
    to='richardiii@shakespeare.lit/desktop'
    type='groupchat'>
  <subject>Made glorious summer by this Sun of York!</subject>
</message>
```

If there is no subject set, the room MUST return an empty <subject/> element.

Example 48. No Subject

```
<message
    from='england@games.shakespeare.lit/sirwilliam'
    id='F437C672-D438-4BD3-9BFF-091050D32EE2'
    to='richardiii@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 MMOG 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 an MMOG room.

Table 10: Error Conditions for Entering a 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 game service, and "nick" is the user’s desired nickname within the room:

Example 49. User Seeks to Enter a Room (groupchat 1.0)

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='ng91xs69'
    to='england@games.shakespeare.lit/earl1'/>
```

This behavior can not be distinguished from a presence update from an MMOG-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 MUC specification, it is RECOMMENDED that a service receiving a <presence> without an <game> 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).

Example 50. Service Response to groupchat 1.0 join / non-occupant presence update

```
<presence
    from='england@games.shakespeare.lit/earl1'
    to='harritudur@shakespeare.lit/pda'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'>
      <reason>You are not in the room.</reason>
    </item>
    <status code='110'/>
    <status code='307'/>
    <status code='333'/>
  </game>
</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. Joining a Team

When a user wants to join a free team, he sends the following presence to <room@service>.

Example 51. User Wants To Join a Team

```
<presence
    from='wcatesby@shakespeare.lit/laptop'
    to='england@games.shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <item role='York'/>
  </game>
</presence>
```

After the team has been successfully assigned to the requesting occupant, the service MUST send the new presence to all occupants.

Example 52. Service Sends Changed Occupant's Presence to All Occupant

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <item affiliation='none' team='york'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <item affiliation='none' team='york'/>
  </game>
</presence>
```

Example 53. Service Sends Changed Occupants Presence Back To Occupant

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <item affiliation='none' team='york'/>
  </game>
</presence>
```

If the team is already taken, the service must return the following error.

Example 54. Service Informs User About Team Conflict

```
<presence
    from='england@games.shakespeare.lit'
    to='wcatesby@shakespeare.lit/laptop'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

If the requested team doesn't exist in the match, the service MUST return a not-acceptable error.

7.5. Match Start

After the match is ready to be started (as to be defined by the game protocol), all players MUST send start messages in order to start the game.

The <start/> element in the <message/> stanza MUST contain an element <pc/> defining the name of which player character (PC) the user is going to play in the upcoming match. The field 'name' is the player character's name; this field MUST present and correspond to the user's name. The field 'visiname' is the player character’s visible name; the field is optional and if it is absent then the visible name is equal to the player character’s name. The field 'model' is the player character’s weapon model; the field MUST present because it defines the model of playable characters the user is acting for.

Example 55. Player Sends a Start Message

```
<message
    from='richardiii@shakespeare.lit/desktop'
    to='england@games.shakespeare.lit'
    type='groupchat'>
  <start xmlns='http://jabber.org/protocol/mmog#user'>
    <pc name='richardiii' visiname='Good Lord' model='King'>
  </start>
</message>
```

In order to see what players are ready to start, the service MUST reflect the start message from each player to all players. In the case when identifiers for player characters (pcs) are assigned by the server, then the reflected messages MUST include unique identifiers for every registered player character to be used within the match.

In the case when identifiers for non-player characters (npcs) are assigned by the server, then the reflected start messages MUST include identifiers generated for all non-player characters to be used within the match.

The server MUST include to the reflected message the <pc> element, containing the player started the match. If the server starts a match only after preparing a list of the players requested for a match (Single Battle Match), then the server MUST return to each player the response including this player in the <pc> element.

In the case of continuing matches (Battle Royale Matches) the server MUST reflect the message with the <pc> element containing the last player requested to join the current match, and the <pcs> element containing the list of players currently participating in the match.

The <pcs> element MUST include all players participating in the match and also the player noticed in the <pc> element.

The order the players are listed in the <pcs> element MUST be the same for all players the message is reflected to.

Example 56. Service Reflects the Start Message

```
<message
    from='england@games.shakespeare.lit/king'
    to='richardiii@shakespeare.lit/desktop'
    type='groupchat'>
  <start xmlns='http://jabber.org/protocol/mmog#user'
         jid='bosworth@games.shakespeare.lit'
         name='bosworth'
         battlefield='Valley'
         match='Single Battle'>
    <pc jid='richardiii@shakespeare.lit/desktop'
        chid='h7ns81g'
        name='richardiii'
        visiname='Good Lord'
        model='King'
        team='York'
        position='0'>

    </pc>
    <pcs>
      <pc jid='richardiii@shakespeare.lit/desktop'
          chid='h7ns81g'
          name='richardiii'
          visiname='Good Lord'
          model='King'
          team='York'
          position='0'>
      <pc jid='wcatesby@shakespeare.lit/laptop'
          chid='lx09df27'
          name='wcatesby'
          visiname='Good Catholic'
          model='Councillor'
          team='York'
          position='1'>

      </pc>
    </pcs>
    <npcs>
      <npc chid='d72d91y'
           name='House01'
           visiname='Watchtower'
           model='Building'
           team='York'
           position='0'>

      </npc>
    </npcs>
  </start>
</message>

<message
    from='england@games.shakespeare.lit/king'
    to='wcatesby@shakespeare.lit/laptop'
    type='groupchat'>
  <start xmlns='http://jabber.org/protocol/mmog#user'
        jid='bosworth@games.shakespeare.lit'
        name='bosworth'
        battlefield='Valley'
        match='Single Battle'>
    <pc jid='wcatesby@shakespeare.lit/laptop'
```

The following table describes the attributes of the game match started on the server and represented by the <start> element returned from the server.

Table 11: Attributes of Game Matches

Element	Meaning
jid	A Jabber Identifier (JID) assigned by the game server to the game match.
name	The game match’s name corresponding its JID.
battlefield	The game match’s battlefield type representing the class of game battlefield the user is going to play on.
match	The game match’s type representing the class of game matches the user is going to play.

The following table describes the attributes of player characters.

Table 12: Attributes of Player Characters

Element	Meaning
jid	A Jabber Identifier (JID) assigned by the game server to the player character.
chid	A Character Identifier (CHID) assigned by the game server to the player character.
name	The player character’s name corresponding to the user’s name.
visiname	The player character’s visible name. The field is optional and if it is absent then the visible name is equal to the player character’s name.
model	The player character’s model representing the class of playable characters the user is acting for.
team	A game team assigned by the game server to the player character.
position	The index of the player character’s disposition on the battlefield during the match starting.

The following table describes the attributes of non-player characters.

Table 13: Attributes of Non-Player Characters

Element	Meaning
chid	A Character Identifier (CHID) assigned by the game server to the non-player character.
name	The non-player character's name.
visiname	The non-player character's visible name. The field is optional and if it is absent then the visible name is equal to the name.
model	The non-player character's model representing the class of non-playable characters.
team	A game team assigned by the game server to the non-player character.
position	The index of the non-player character's disposition on the battlefield during the match starting.

If the match is not ready, the service MUST send the following error.

Example 57. Service Informs Player that the Match is Not Ready

```
<message
    from='england@games.shakespeare.lit'
    to='richardiii@shakespeare.lit/desktop'
    type='error'>
  <start xmlns='http://jabber.org/protocol/mmog#user'>
    <pc name='richardiii' visiname='Good Lord' model='King'>
  </start>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</message>
```

After the service received messages from all players, it MUST update the match status from inactive to active by a presence broadcast to all occupants. If the owner changes the configuration or roles change after a player sent his message and before the service sends presence broadcast indicating the game status active, the player MUST send the message again.

Example 58. Service Broadcasts the Start Message to All

```
<presence
    from='england@games.shakespeare.lit'
    to='elizabeth@shakespeare.lit/tablet'
    type='groupchat'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <pc
      jid='elizabeth@shakespeare.lit/tablet'
      chid='zb8q41f4'
      name='elizabeth'
      visiname='White Rose'
      model='Queen Consort'
      team='York'
      position='2'>
    <status>active</status>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      <x xmlns='jabber:x:data' type='submit'>
        ...
      </x>
    </state>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit'
    to='wcatesby@shakespeare.lit/laptop'
    type='groupchat'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <pc jid='wcatesby@shakespeare.lit/laptop'
        chid='lx09df27'
        name='wcatesby'
        visiname='Good Catholic'
        model='Councillor'
        team='York'
        position='1'>
    <status>active</status>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      <x xmlns='jabber:x:data' type='submit'>
        ...
      </x>
    </state>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit'
    to='richardiii@shakespeare.lit/desktop'
    type='groupchat'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <pc
      jid='richardiii@shakespeare.lit/desktop'
      chid='h7ns81g'
      name='richardiii'
      visiname='Good Lord'
      model='King'
      team='York'
      position='0'>
    <status>active</status>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      <x xmlns='jabber:x:data' type='submit'>
```

Note that the spectator "elizabeth" receives the start presence, too.

7.6. Move in Match

In order to make a move in a match, a <message/> stanza MUST be send to <room@service/nick> containing a <turn/> element qualified by 'http://jabber.org/protocol/mmog#user' namespace. Game protocols SHOULD place their own elements defining the turn inside the <turn/> element.

A move is defined in the element of the same name <move> which MUST be included within the element <pc> of an appropriate playing character.

Example 59. Occupant Sends a Move in a Game Turn

```
<message
    from='richardiii@shakespeare.lit/desktop'
    id='hysf1v37'
    to='bosworth@games.shakespeare.lit'
    type='groupchat'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'
        tuid='dyb12s6'>
    <pc jid='richardiii@shakespeare.lit/desktop'
        chid='h7ns81g'
        name='richardiii'
        visiname='Good Lord'
        model='King'
        team='York'
        position='0'>
      <moves>
        <move xmlns='http://jabber.org/protocol/mmog#user'
              node='Chevalier'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
        </move>
        <move xmlns='http://jabber.org/protocol/mmog#user'
              node='Hull'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
        </move>
        <move xmlns='http://jabber.org/protocol/mmog#user'
              node='Head'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
        </move>
      </moves>
    </pc>
    <npc chid='e79ayaa'
         name='Tower01'
         visiname='Donjon'
         model='Siegetower'
         team='York'
         position='0'>
      <moves>
        <move xmlns='http://jabber.org/protocol/mmog#user'
              node='Tower01'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
```

The element <move> CAN include internal elements as the following ones in the table below. Every element contains attributes defining physical conditions of the player character in the game space.

Table 14: Internal Elements of a Move

Element	Meaning
location	The player character's location in the game space.
rotation	The player character's rotation in the game space.
view	The player character's viewing direction.
walk	The player character's walking direction.
linear	The player character's linear velocity.
angular	The player character's angular velocity.

The element <move> MUST include the field 'node' defining the identifier (name) of some part of the player character.

A service MUST validate the player's move before passing it to the occupants. If the turn is invalid, as defined by the game protocol, the service MUST return an error and kick the player unless the player is the owner of the room, in which case he SHOULD lose his game role.

Example 60. Service Informs Player About an Invalid Turn

```
<message
    from='bosworth@games.shakespeare.lit'
    id='hysf1v37'
    to='richardiii@shakespeare.lit/desktop'
    type='error'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'>
    <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
  </turn>
  <error type='cancel'>
    <undefined-condition xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <invalid-turn xmlns='http://jabber.org/protocol/mmog'/>
  </error>
</message>

<presence
    from='bosworth@games.shakespeare.lit/buckingham'
    to='richardiii@shakespeare.lit/desktop'
    type='unavailable'>
  <invalid-turn xmlns='http://jabber.org/protocol/mmog#user'/>
</presence>
```

If a spectator sends a turn the service MUST return the following error.

Example 61. Service Denies Turn

```
<message
    from='bosworth@games.shakespeare.lit/buckingham'
    id='hysf1v37'
    to='elizabeth@shakespeare.lit/tablet'
    type='error'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'>
    <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
  </turn>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</message>
```

If a player sends a turn while the match status is 'inactive' or 'paused' the service MUST send this error:

Example 62. Service Denies Turn Because of Match Status

```
<message
    from='richardiii@shakespeare.lit/desktop'
    id='hysf1v37'
    to='bosworth@games.shakespeare.lit'
    type='error'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'>
    <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
  </turn>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</message>
```

If the move is valid, the service MUST distribute the turn. The occupants, who receive the turn are defined by the game protocol. However, the turn MUST be reflected to the sender.

Example 63. Service Reflects Turn to All Occupants

```
<message
    from='bosworth@games.shakespeare.lit/buckingham'
    id='hysf1v37'
    to='wcatesby@shakespeare.lit/laptop'
    type='groupchat'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'>
    <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
  </turn>
</message>

<message
    from='bosworth@games.shakespeare.lit/buckingham'
    id='hysf1v37'
    to='elizabeth@shakespeare.lit/tablet'
    type='groupchat'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'>
    <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
  </turn>
</message>

<message
    from='bosworth@games.shakespeare.lit/buckingham'
    id='hysf1v37'
    to='richardiii@shakespeare.lit/desktop'
    type='groupchat'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'>
    <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
  </turn>
</message>

<message
    from='bosworth@games.shakespeare.lit/buckingham'
    id='hysf1v37'
    to='harritudur@shakespeare.lit/pda'
    type='groupchat'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'>
    <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
  </turn>
</message>
```

7.7. Shot in Match

In order to make a shot in a match, a <message/> stanza MUST be send to <room@service/nick> containing a <turn/> element which contains a <shot> element specifying the shot’s firepoint.

Example 64. Occupant sends a Shoot in a Game Turn

```
<message
    from='richardiii@shakespeare.lit/desktop'
    id='daf90m1'
    to='bosworth@games.shakespeare.lit'
    type='groupchat'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'
        tuid='aa7y4m2'>
    <pc jid='richardiii@shakespeare.lit/desktop'
        chid='h7ns81g'
        name='richardiii'
        visiname='Good Lord'
        model='King'
        team='York'
        position='0'>
      <shots>
        <shot xmlns='http://jabber.org/protocol/mmog#user'
              node='Turmkanone'
              weapon='KwK42L70'
              ammo='Pzgr39_42'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
        </shot>
      </shots>
    </pc>
    <npc chid='c12y325'
         name='Cannon01'
         visiname='Dragon'
         model='Cannon'
         team='York'
         position='0'>
      <shots>
        <shot xmlns='http://jabber.org/protocol/mmog#user'
              node='Cannon01'
              weapon='38cmS.K.C/34'
              ammo='PsgrL/4,4mBdZ'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
        </shot>
      </shots>
    </npc>
  </turn>
</message>
```

The element <shot> CAN include internal elements as the following ones in the table below. Every element contains attributes defining physical conditions of the player character in the game space.

Table 15: Internal Elements of a Shot

Element	**Meaning**
location	The player character’s location in the game space.
rotation	The player character’s rotation in the game space.
view	The player character’s viewing direction.
walk	The player character’s walking direction.
linear	The player character’s linear velocity.
angular	The player character’s angular velocity.

The element <shot> MUST include the field 'node' defining the identifier (name) of some part of the player character.

7.8. Loss in Match

In order to make a loss in a match, a <message/> stanza MUST be send to <room@service/nick> containing a <turn/> element qualified by 'http://jabber.org/protocol/mmog#user' namespace. Game protocols SHOULD place own elements defining the turn inside the <turn/> element.

A loss is defined in the element of the same name <loss> which MUST be included within the element <pc> of an appropriate playing character.

Example 65. Occupant Sends a Loss in a Game Turn

```
<message
    from='richardiii@shakespeare.lit/desktop'
    id='daf90m1'
    to='bosworth@games.shakespeare.lit'
    type='groupchat'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'
        tuid='aa7y4m2'>
    <pc jid='richardiii@shakespeare.lit/desktop'
        chid='h7ns81g'
        name='richardiii'
        visiname='Good Lord'
        model='King'
        team='York'
        position='0'>
      <losses>
        <loss xmlns='http://jabber.org/protocol/mmog#user'
              node='Tank'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
          <power value=''/>
        </loss>
        <loss xmlns='http://jabber.org/protocol/mmog#user'
              node='Hull'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
          <power value=''/>
        </loss>
      </losses>
    </pc>
    <npc chid='d72d91y'
         name='House01'
         visiname='Watchtower'
         model='Building'
         team='York'
         position='0'>
      <losses>
        <loss xmlns='http://jabber.org/protocol/mmog#user'
              node='House01'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
          <power value=''/>
        </loss>
      </losses>
    </npc>
  </turn>
</message>
```

The element <loss> CAN include internal elements as the following ones in the table below. Every element contains attributes defining physical conditions of the player character in the game space.

Table 16: Internal Elements of a Loss

Element	Meaning
location	The player character’s location in the game space.
rotation	The player character’s rotation in the game space.
view	The player character’s viewing direction.
walk	The player character’s walking direction.
linear	The player character’s linear velocity.
angular	The player character’s angular velocity.

The element <loss> MUST include the field 'node' defining the identifier (name) of some part of the player character.

7.9. Repair in Match

In order to make a repair in a match, a <message/> stanza MUST be send to <room@service/nick> containing a <turn/> element qualified by 'http://jabber.org/protocol/mmog#user' namespace. Game protocols SHOULD place own elements defining the turn inside the <turn/> element.

A repair is defined in the element of the same name <repair> which MUST be included within the element <pc> of an appropriate playing character.

Example 66. Occupant Sends a Repair in a Game Turn

```
<message
    from='richardiii@shakespeare.lit/desktop'
    id='daf90m1'
    to='bosworth@games.shakespeare.lit'
    type='groupchat'>
  <turn xmlns='http://jabber.org/protocol/mmog#user'
        tuid='aa7y4m2'>
    <pc jid='richardiii@shakespeare.lit/desktop'
        chid='h7ns81g'
        name='richardiii'
        visiname='Good Lord'
        model='King'
        team='York'
        position='0'>
      <repairs>
        <repair xmlns='http://jabber.org/protocol/mmog#user'
              node='Tank'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
          <power value=''/>
        </repair>
        <repair xmlns='http://jabber.org/protocol/mmog#user'
              node='Hull'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
          <power value=''/>
        </repair>
      </repairs>
    </pc>
    <npc chid='d72d91y'
         name='House01'
         visiname='Watchtower'
         model='Building'
         team='York'
         position='0'>
      <repairs>
        <repair xmlns='http://jabber.org/protocol/mmog#user'
              node='House01'>
          <location x='' y='' z=''/>
          <rotation x='' y='' z='' w=''/>
          <view x='' y='' z=''/>
          <walk x='' y='' z=''/>
          <linear x='' y='' z=''/>
          <angular x='' y='' z=''/>
          <power value=''/>
        </repair>
      </repairs>
    </npc>
  </turn>
</message>
```

The element <repair> CAN include internal elements as the following ones in the table below. Every element contains attributes defining physical conditions of the player character in the game space.

Table 17: Internal Elements of a Repair

Element	Meaning
location	The player character’s location in the game space.
rotation	The player character’s rotation in the game space.
view	The player character’s viewing direction.
walk	The player character’s walking direction.
linear	The player character’s linear velocity.
angular	The player character’s angular velocity.

The element <repair> MUST include the field 'node' defining the identifier (name) of some part of the player character.

7.10. Match Resignation

If a client wants to resign, he sends the following.

Example 67. User Resigns

```
<presence
    from='richardiii@shakespeare.lit/desktop'
    to='england@games.shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <item role='none'/>
  </game>
</presence>
```

Afterwards, the service decides whether to cancel or pause the match based on the game specification.

7.11. Match Termination

The game protocol respectively the game protocol implementation decides when a match is over. In the case of game termination, the service MUST notify every player through updated presence including the resulting final state.

Example 68. Service Sends Termination Broadcast to All Players

```
<presence
    from='england@games.shakespeare.lit'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <status>inactive</status>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      <won>York</won>
    </state>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit'
    to='elizabeth@shakespeare.lit/tablet'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <status>inactive</status>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      <won>York</won>
    </state>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <status>inactive</status>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      <won>York</won>
    </state>
  </game>
</presence>
```

7.12. 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.

Example 69. Occupant Sends a Message to All Occupants

```
<message
    from='harritudur@shakespeare.lit/pda'
    id='hysf1v37'
    to='england@games.shakespeare.lit'
    type='groupchat'>
  <body>How far into the morning is it, lords?</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.

Example 70. Service Reflects Message to All Occupants

```
<message
    from='england@games.shakespeare.lit/earl1'
    id='hysf1v37'
    to='richardiii@shakespeare.lit/desktop'
    type='groupchat'>
  <body>How far into the morning is it, lords?</body>
</message>

<message
    from='england@games.shakespeare.lit/earl1'
    id='hysf1v37'
    to='wcatesby@shakespeare.lit/laptop'
    type='groupchat'>
  <body>How far into the morning is it, lords?</body>
</message>

<message
    from='england@games.shakespeare.lit/earl1'
    id='hysf1v37'
    to='thomasstanley@shakespeare.lit/cell'
    type='groupchat'>
  <body>How far into the morning is it, lords?</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).

> Note: the requirement to reflect the 'id' attribute was added in version 1.31 of the XEP MUC. Servers following that specification SHOULD advertise that with a disco info feature of 'http://jabber.org/protocol/mmog#stable_id' on both the service domain and on individual MMOGs, 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.

7.13. 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 "mmog#roomconfig_allowpm" room configuration option.

To allow for proper synchronization of these messages to the user's other clients by Message Carbons (XEP-0280), the sending client SHOULD add an <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace to the message.

> Note: because this requirement was only added in revision 1.28 of the XEP MUC, receiving entities MUST NOT rely on the existence of the <game/> element on private messages for proper processing.

Example 71. Occupant Sends Private Message

```
<message
    from='wcatesby@shakespeare.lit/laptop'
    id='hgn27af1'
    to='england@games.shakespeare.lit/king'
    type='chat'>
  <body>Rescue, fair lord, or else the day is lost!</body>
  <game xmlns='http://jabber.org/protocol/mmog#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 <game/> element if the message does not contain it already.

Example 72. Recipient Receives the Private Message

```
<message
    from='england@games.shakespeare.lit/sirwilliam'
    id='hgn27af1'
    to='richardiii@shakespeare.lit/desktop'
    type='chat'>
  <body>A horse! a horse! my kingdom for a horse!</body>
  <game xmlns='http://jabber.org/protocol/mmog#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:

Example 73. Occupant Attempts to Send a Message of Type "Groupchat" to a Particular Occupant

```
<message
    from='wcatesby@shakespeare.lit/laptop'
    id='bx71f29k'
    to='england@games.shakespeare.lit/king'
    type='groupchat'>
  <body>Rescue, fair lord, or else the day is lost!</body>
</message>

<message
    from='england@games.shakespeare.lit/king'
    id='bx71f29k'
    to='wcatesby@shakespeare.lit/laptop'
    type='error'>
  <body>Rescue, fair lord, or else the day is lost!</body>
  <error by='england@games.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.

7.14. Changing Nickname

A common feature of game rooms is the ability for an occupant to change his or her nickname within the room. In MMOG 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).

Example 74. Occupant Changes Nickname

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='ifd1c35'
    to='england@games.shakespeare.lit/henryvii'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
</presence>
```

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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace:

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

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

Example 75. Service Updates Nick

```
<presence
    from='england@games.shakespeare.lit/earl1'game
    id='5C1B95B3-7CCC-4422-A952-8885A050BDE9'
    to='richardiii@shakespeare.lit/desktop'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='harritudur@shakespeare.lit/pda'
          nick='henryvii'
          role='participant'/>
    <status code='303'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/earl1'
    id='B0E6ABD5-575D-42F0-8242-569004D88F73'
    to='wcatesby@shakespeare.lit/laptop'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='harritudur@shakespeare.lit/pda'
          nick='henryvii'
          role='participant'/>
    <status code='303'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/earl1'
    id='DC352437-C019-40EC-B590-AF29E879AF98'
    to='harritudur@shakespeare.lit/pda'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='harritudur@shakespeare.lit/pda'
          nick='henryvii'
          role='participant'/>
    <status code='303'/>
    <status code='110'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/henryvii'
    id='19E41EB3-3F4C-444F-8A1B-713A8860980C'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='harritudur@shakespeare.lit/pda'
          role='participant'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/henryvii'
    id='79225CEA-F610-49BE-9B97-FEFA8737185B'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
```

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

Example 76. Occupant Changes Nickname, Modified by Service

```
<presence
    from='harritudur@shakespeare.lit/pda'
    id='nx6z2v5'
    to='england@games.shakespeare.lit/henryvii'/>

<presence
    from='england@games.shakespeare.lit/henryvii'
    id='D0E2B666-3373-42C9-B726-D52C40A48383'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='harritudur@shakespeare.lit/pda'
          role='participant'/>
    <status code='110'/>
    <status code='210'/>
  </game>
</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:

Example 77. Service Denies Nickname Change Because of Nick Conflict

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='ifd1c35'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.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:

Example 78. Service Denies Nickname Change Because Roomnicks Are Locked Down

```
<presence
    from='england@games.shakespeare.lit/earl1'
    id='ifd1c35'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
  <error by='england@games.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.

7.15. 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 "earl1|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>.

Example 79. Occupant Changes Availability Status

```
<presence
    from='wcatesby@shakespeare.lit/laptop'
    id='kr7v143h'
    to='england@games.shakespeare.lit/henryvii'>
  <show>xa</show>
  <status>Ay, my good lord.</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:

Example 80. Service Passes Along Changed Presence to All Occupants

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    id='86E11ABF-26BC-46F1-AD3B-F5E54F3C1EE5'
    to='richardiii@shakespeare.lit/desktop'>
  <show>xa</show>
  <status>Ay, my good lord.</status>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit/laptop'
          role='moderator'/>
  </game>
</presence>

[ ... ]
```

7.16. 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) 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.

7.16.1 Direct Invitation

A method for sending a direct invitation (not mediated by the room itself) is defined in Direct MUC Invitations (XEP-0249). 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).

7.16.2. Mediated Invitation

It can be useful to invite another user to a room in which one is an occupant. To send a mediated invitation, an MMOG 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"):

Example 81. Occupant Sends a Mediated Invitation

```
<message
    from='wcatesby@shakespeare.lit/laptop'
    id='nzd143v8'
    to='england@games.shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <invite to='whastings@shakespeare.lit'>
      <reason>
        God keep your lordship in that gracious mind!
      </reason>
    </invite>
  </game>
</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):

Example 82. Room Sends Invitation to Invitee on Behalf of Invitor

```
<message
    from='england@games.shakespeare.lit'
    id='nzd143v8'
    to='whastings@shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <invite from='wcatesby@shakespeare.lit/laptop'>
      <reason>
        God keep your lordship in that gracious mind!
      </reason>
    </invite>
    <password>dieuetmondroit</password>
  </game>
</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 <game 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:

Example 83. Invitee Declines Invitation

```
<message
    from='whastings@shakespeare.lit/smartphone'
    id='jk2vs61v'
    to='england@games.shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <decline to='sirwilliam@shakespeare.lit'>
      <reason>
        I'll wait upon your lordship.
      </reason>
    </decline>
  </game>
</message>
```

Example 84. Room Informs Invitor that Invitation Was Declined

```
<message
    from='england@games.shakespeare.lit'
    id='jk2vs61v'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <decline from='whastings@shakespeare.lit'>
      <reason>
        I'll wait upon your lordship.
      </reason>
    </decline>
  </game>
</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.

7.17. Converting a One-to-One Game Into a Multi-User Game

Sometimes it is desirable to convert a one-to-one game into a multi-user game. The process is as follows.

First, two users begin a one-to-one game.

Example 85. A One-to-One Game

```
<message
    from='richardiii@shakespeare.lit/desktop'
    id='mjs51f36'
    to='wcatesby@shakespeare.lit/laptop'
    type='chat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <start xmlns='http://jabber.org/protocol/mmog#user'/>
</message>

<message
    from='wcatesby@shakespeare.lit/laptop'
    id='l9ij1f3h'
    to='richardiii@shakespeare.lit/desktop'
    type='chat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <start xmlns='http://jabber.org/protocol/mmog#user'/>
</message>
```

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

1. Creates a new multi-user game room
2. Sends history of the one-to-one game 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 game 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 game and the multi-user game.

Example 86. Continuing the Game I: User Creates Room

```
<presence
    from='richardiii@shakespeare.lit/desktop'
    to='england@games.shakespeare.lit/king'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
</presence>

<presence
    from='england@games.shakespeare.lit/king'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='owner' role='moderator'/>
    <status code='110'/>
  </game>
</presence>
```

Example 87. Continuing the Game II: Owner Sends History to Room

```
<message
    from='richardiii@shakespeare.lit/desktop'
    id='b4va73n0'
    to='england@games.shakespeare.lit'
    type='groupchat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <start xmlns='http://jabber.org/protocol/mmog#user'/>
  <delay xmlns='urn:xmpp:delay'
      from='richardiii@shakespeare.lit/desktop'
      stamp='2019-09-09T09:26:37Z'>
    <turn xmlns='http://jabber.org/protocol/mmog#user'>
      <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
    </turn>
  </delay>
</message>

<message
    from='richardiii@shakespeare.lit/desktop'
    id='i4hs759k'
    to='england@games.shakespeare.lit'
    type='groupchat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <start xmlns='http://jabber.org/protocol/mmog#user'/>
  <delay xmlns='urn:xmpp:delay'
      from='richardiii@shakespeare.lit/desktop'
      stamp='2019-09-09T09:26:42Z'>
    <turn xmlns='http://jabber.org/protocol/mmog#user'>
      <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='bishop'
          col='e'
          row='4'/>
    </turn>
  </delay>
</message>
```

Note: Use of the Delayed Delivery (XEP-0203) protocol enables the room creator to specify the datetime of each message from the one-to-one game 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 game history before inviting additional users to the room, and also send as history any messages appearing in the one-to-one game 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 game history messages received from the room owner.

Example 88. Continuing the Discussion III: Owner Sends Invitations, Including Continue Flag

```
<message
    from='richardiii@shakespeare.lit/desktop'
    id='gl3s85n7'
    to='england@games.shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <invite to='wcatesby@shakespeare.lit/laptop'>
      <reason>This battle needs both sirwilliam and earl1.</reason>
      <continue thread='e0ffe42b28561960c6b12b944a092794b9683a38'/>
    </invite>
    <invite to='harritudur@shakespeare.lit/pda'>
      <reason>This battle needs both sirwilliam and earl1.</reason>
      <continue thread='e0ffe42b28561960c6b12b944a092794b9683a38'/>
    </invite>
  </game>
</message>
```

Note: Since the inviter’s client knows the full JID of the person with whom the inviter was having a one-to-one game, it SHOULD include the full JID (rather than the bare JID) in its invitation to that user.

The invitations are delivered to the invitees:

Example 89. Invitations Delivered

```
<message
    from='england@games.shakespeare.lit/king'
    id='DB0414CB-AFBA-407E-9DE3-0E014E84860F'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <invite from='king@shakespeare.lit'>
      <reason>This battle needs both sirwilliam and earl1.</reason>
      <continue thread='e0ffe42b28561960c6b12b944a092794b9683a38'/>
    </invite>
  </game>
</message>

<message
    from='england@games.shakespeare.lit/king'
    id='89028D79-AB4C-44C0-BE81-B07607C2F4C2'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <invite from='king@shakespeare.lit'>
      <reason>This battle needs both sirwilliam and earl1.</reason>
      <continue thread='e0ffe42b28561960c6b12b944a092794b9683a38'/>
    </invite>
  </game>
</message>
```

When the client being used by <wcatesby@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 game window into a multi-user gaming window:

Example 90. Invitee Accepts Invitation, Joins Room, and Receives Presence and History

```
<presence
    from='richardiii@shakespeare.lit/desktop'
    to='england@games.shakespeare.lit/sirwilliam'>
  <game xmlns='http://jabber.org/protocol/mmog'/>
</presence>

<presence
    from='england@games.shakespeare.lit/king'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='owner' role='moderator'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member' role='participant'/>
  </game>
</presence>

<message
    from='england@games.shakespeare.lit'
    id='67268D36-100C-457D-A769-8A3663BD1949'
    to='wcatesby@shakespeare.lit/laptop'
    type='groupchat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <start xmlns='http://jabber.org/protocol/mmog#user'/>
  <delay xmlns='urn:xmpp:delay'
      from='richardiii@shakespeare.lit/desktop'
      stamp='2019-09-09T09:26:37Z'>
    <turn xmlns='http://jabber.org/protocol/mmog#user'>
      <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='queen'
          col='d'
          row='2'/>
    </turn>
  </delay>
</message>

<message
    from='england@games.shakespeare.lit'
    id='367DCF6B-0CB4-482D-A142-C0B9E08016B5'
    to='wcatesby@shakespeare.lit/laptop'
    type='groupchat'>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
  <start xmlns='http://jabber.org/protocol/mmog#user'/>
  <delay xmlns='urn:xmpp:delay'
      from='richardiii@shakespeare.lit/desktop'
      stamp='2019-09-09T09:26:42Z'>
    <turn xmlns='http://jabber.org/protocol/mmog#user'>
      <play xmlns='http://jabber.org/protocol/mmog/dga'
          suit='bishop'
          col='e'
          row='4'/>
    </turn>
  </delay>
</message>
```

7.18. 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):

Example 91. User Requests Registration Requirements

```
<iq from='harritudur@shakespeare.lit/pda'
    id='jw81b36f'
    to='england@games.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.

Example 92. Room Does Not Exist

```
<iq from='england@games.shakespeare.lit'
    id='jw81b36f'
    to='harritudur@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.

Example 93. User Is Not Allowed to Register

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

If the user is already registered, as described in In-Band Registration (XEP-0077) 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.

Example 94. User Is Already Registered

```
<iq from='england@games.shakespeare.lit'
    id='jw81b36f'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='jabber:iq:register'>
    <registered/>
    <username>earl1</username>
  </query>
</iq>
```

Otherwise, the room MUST then return a Data Form to the user (as described in Data Forms (XEP-0004)). 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/mmog#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:

Example 95. Service Returns Registration Form

```
<iq from='england@games.shakespeare.lit'
    id='jw81b36f'
    to='harritudur@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>The Wars of the Roses Registration</title>
      <instructions>
        Please provide the following information
        to register with this room.
      </instructions>
      <field
          type='hidden'
          var='FORM_TYPE'>
        <value>http://jabber.org/protocol/mmog#register</value>
      </field>
      <field
          label='Given Name'
          type='text-single'
          var='mmog#register_first'>
        <required/>
      </field>
      <field
          label='Family Name'
          type='text-single'
          var='mmog#register_last'>
        <required/>
      </field>
      <field
          label='Desired Nickname'
          type='text-single'
          var='mmog#register_roomnick'>
        <required/>
      </field>
      <field
          label='Your URL'
          type='text-single'
          var='mmog#register_url'/>
      <field
          label='Email Address'
          type='text-single'
          var='mmog#register_email'/>
      <field
          label='FAQ Entry'
          type='text-multi'
          var='mmog#register_faqentry'/>
    </x>
  </query>
</iq>
```

The user SHOULD then submit the form:

Example 96. User Submits Registration Form

```
<iq from='harritudur@shakespeare.lit/pda'
    id='nv71va54'
    to='england@games.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/mmog#register</value>
      </field>
      <field var='mmog#register_first'>
        <value>Harri</value>
      </field>
      <field var='mmog#register_last'>
        <value>Tudur</value>
      </field>
      <field var='mmog#register_roomnick'>
        <value>earl1</value>
      </field>
      <field var='mmog#register_url'>
        <value>http://thewarsoftheroses.net/harri-tudur/</value>
      </field>
      <field var='mmog#register_email'>
        <value>harri.tudur@thewarsoftheroses.net</value>
      </field>
      <field var='mmog#register_faqentry'>
        <value>Henry Tudor, the Earl of Richmond.</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:

Example 97. Room Returns Conflict Error to User

```
<iq from='england@games.shakespeare.lit'
    id='nv71va54'
    to='harritudur@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:

Example 98. Room Returns Service Unavailable Error to User

```
<iq from='england@games.shakespeare.lit'
    id='nv71va54'
    to='harritudur@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:

Example 99. Room Returns Service Bad Request Error to User

```
<iq from='england@games.shakespeare.lit'
    id='nv71va54'
    to='harritudur@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:

Example 100. Room Informs User that Registration Request Has Been Processed

```
<iq from='england@games.shakespeare.lit'
    id='nv71va54'
    to='harritudur@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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "member".

Example 101. Service Sends Notice of Membership to All Occupants

```
<presence
    from='england@games.shakespeare.lit/earl1'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='harritudur@shakespeare.lit/pda'
          role='participant'/>
  </game>
</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).

7.19. 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.

7.20. 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".

Example 102. User Requests Reserved Nickname

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

It is OPTIONAL for a multi-user game 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 "game/multi-user"):

Example 103. Room Returns Nickname

```
<iq from='england@games.shakespeare.lit'
    id='getnick1'
    to='harritudur@shakespeare.lit/pda'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='x-roomuser-item'>
    <identity
        category='game'
        name='earl1'
        type='multi-user'/>
  </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)).

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.

7.21. 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 "mmog#role" field with a value of "participant".

Example 104. Occupant Requests Voice

```
<message from='harritudur@shakespeare.lit/pda'
         id='yd53c486'
         to='england@games.shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='FORM_TYPE'>
        <value>http://jabber.org/protocol/mmog#request</value>
      </field>
      <field var='mmog#role'
          type='list-single'
          label='Requested role'>
        <value>participant</value>
      </field>
    </x>
  </game>
</message>
```

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

7.22. Exiting a Room

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

Example 105. Occupant Exits a Room

```
<presence
    from='harritudur@shakespeare.lit/pda'
    to='england@games.shakespeare.lit/earl1'
    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":

Example 106. Service Sends Self-Presence Related to Departure of Occupant

```
<presence
    from='england@games.shakespeare.lit/earl1'
    to='harritudur@shakespeare.lit/pda'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
        jid='harritudur@shakespeare.lit/pda'
        role='none'/>
    <status code='110'/>
  </game>
</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:

Example 107. Service Sends Presence Related to Departure of Occupant

```
<presence
    from='england@games.shakespeare.lit/earl1'
    to='harritudur@shakespeare.lit/pda'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
        jid='harritudur@shakespeare.lit/pda'
        role='none'/>
    <status code='110'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/earl1'
    to='richardiii@shakespeare.lit/desktop'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
        jid='harritudur@shakespeare.lit/pda'
        role='none'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/earl1'
    to='wcatesby@shakespeare.lit/laptop'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
        jid='harritudur@shakespeare.lit/pda'
        role='none'/>
  </game>
</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:

Example 108. Custom Exit Message

```
<presence
    from='wcatesby@shakespeare.lit/laptop'
    to='england@games.shakespeare.lit/henryvii'
    type='unavailable'>
  <status>Now civil wounds are stopped, peace lives again.</status>
</presence>
```

Normal presence stanza generation rules apply as defined in XMPP IM, 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 without sending unavailable presence, the user's server is responsible for sending unavailable presence on behalf of the user (in accordance with RFC 6121).

> 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 MMOG 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/mmog#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 game 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 might be allowed to change the subject, as controlled by the "mmog#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.

Example 109. Moderator Changes Subject

```
<message
    from='wcatesby@shakespeare.lit/laptop'
    id='lh2bs617'
    to='england@games.shakespeare.lit'
    type='groupchat'>
  <subject>Made glorious summer by this Sun of York!</subject>
</message>
```

The MMOG 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:

Example 110. Service Informs All Occupants of Subject Change

```
<message
    from='england@games.shakespeare.lit/sirwilliam'
    id='5BCE07C5-0729-4353-A6A3-ED9818C9B498'
    to='richardiii@shakespeare.lit/desktop'
    type='groupchat'>
  <subject>Made glorious summer by this Sun of York!</subject>
</message>

[ ... ]
```

As explained under , when a new occupant joins the room the room SHOULD include the last subject change after the discussion history.

An MMOG client that receives such a message MAY choose to display an in-room message, such as the following:

Example 111. Client Displays Room Subject Change Message

```
* sirwilliam has changed the subject to: Made glorious summer by this Sun of York!
```

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:

Example 112. Service Returns Error Related to Unauthorized Subject Change

```
<message
    from='england@games.shakespeare.lit/earl1'
    id='lh2bs617'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <subject>Thou offspring of the house of Lancaster!</subject>
  <error by='england@games.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>").

Example 113. Moderator Sets Empty Subject

```
<message
    from='wcatesby@shakespeare.lit/laptop'
    id='uj3bs61g'
    to='england@games.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".

Example 114. Moderator Kicks Occupant

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='kick1'
    to='buckingham@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='buckingham' role='none'>
      <reason>Out on you, owls!</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.

Example 115. Service Removes Kicked Occupant

```
<presence
    from='richardiii@shakespeare.lit/desktop'
    to='buckingham@shakespeare.lit/notebook'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'>
      <actor nick='Buckingham'/>
      <reason>Out on you, owls!</reason>
    </item>
    <status code='110'/>
    <status code='307'/>
  </game>
</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. *(N.B.)*

After removing the kicked occupant(s), the service MUST then inform the moderator of success:

Example 116. Service Informs Moderator of Success

```
<iq from='buckingham@games.shakespeare.lit'
    id='kick1'
    to='richardiii@shakespeare.lit/desktop'
    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.

Example 117. Service Informs Remaining Occupants

```
<presence
    from='richardiii@shakespeare.lit/desktop'
    to='wcatesby@shakespeare.lit/laptop'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'/>
    <status code='307'/>
  </game>
</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:

Example 118. Service Returns Error on Attempt to Kick User With Higher Affiliation

```
<iq from='england@games.shakespeare.lit'
    id='kicktest'
    to='wcatesby@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".

Example 119. Moderator Grants Voice to a Visitor

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='voice1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='messenger'
          role='participant'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL:

Example 120. Moderator Grants Voice to a Visitor (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='voice1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='messenger'
          role='participant'>
      <reason>Well, go muster man.</reason>
    </item>
  </query>
</iq>
```

The service MUST then inform the moderator of success:

Example 121. Service Informs Moderator of Success

```
<iq from='england@games.shakespeare.lit'
    id='voice1'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "participant".

Example 122. Service Sends Notice of Voice to All Occupants

```
<presence
    from='england@games.shakespeare.lit/king'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          nick='messenger'
          role='participant'>
      <reason>Well, go muster man.</reason>
    </item>
  </game>
</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":

Example 123. Moderator Revokes Voice from a Participant

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='voice2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='messenger'
          role='visitor'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL:

Example 124. Moderator Revokes Voice from a Visitor (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='voice2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='messenger'
        role='visitor'>
      <reason>Take that, until thou bring me better news.</reason>
    </item>
  </query>
</iq>
```

The service MUST then inform the moderator of success:

Example 125. Service Informs Moderator of Success

```
<iq from='england@games.shakespeare.lit'
    id='voice2'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "visitor".

Example 126. Service Notes Loss of Voice

```
<presence
    from='england@games.shakespeare.lit/king'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
        jid='harritudur@shakespeare.lit/pda'
        role='visitor'/>
  </game>
</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):

Example 127. Service Returns Error on Attempt to Revoke Voice from an Admin, Owner, or User with a Higher Affiliation

```
<iq from='england@games.shakespeare.lit'
    id='voicetest'
    to='wcatesby@shakespeare.lit/laptop'
    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'.

Example 128. Moderator Requests Voice List

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='voice3'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#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:

Example 129. Service Sends Voice List to Moderator

```
<iq from='england@games.shakespeare.lit'
    id='voice3'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit/laptop'
          nick='sirwilliam'
          role='participant'/>
    <item affiliation='none'
          jid='harritudur@shakespeare.lit/pda'
          nick='earl1'
          role='participant'/>
    <item affiliation='none'
          jid='thomasstanley@shakespeare.lit/cell'
          nick='earlofderby'
          role='participant'/>
    <item affiliation='none'
          jid='elizabeth@shakespeare.lit/tablet'
          nick='queenconsort'
          role='none'/>
  </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):

Example 130. Moderator Sends Modified Voice List to Service

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='voice4'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='earl1'
          role='outcast'/>
    <item nick='earlofderby'
          role='outcast'>
      <reason>For, lords, tomorrow is a busy day.</reason>
    </item>
    <item nick='queenconsort'
          role='participant'>
      <reason>So, I am satisfied.</reason>
    </item>
  </query>
</iq>
```

The service MUST then inform the moderator of success:

Example 131. Service Informs Moderator of Success

```
<iq from='england@games.shakespeare.lit'
    id='voice1'
    to='richardiii@shakespeare.lit/desktop'
    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:

Example 132. Service Returns Error on Attempt to Revoke Voice from an Admin, Owner, or User with a Higher Affiliation

```
<iq from='england@games.shakespeare.lit'
    id='voicetest'
    to='wcatesby@shakespeare.lit/laptop'
    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.

Example 133. Voice Request Approval Form

```
<message from='england@games.shakespeare.lit'
         id='approve'
         to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <options>
      <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/mmog#request</value>
        </field>
        <field var='mmog#role'
               type='list-single'
               label='Requested role'>
          <value>participant</value>
        </field>
        <field var='mmog#jid'
               type='jid-single'
               label='User ID'>
          <value>harritudur@shakespeare.lit/pda</value>
        </field>
        <field var='mmog#roomnick'
               type='text-single'
               label='Room Nickname'>
          <value>earl1</value>
        </field>
        <field var='mmog#request_allow'
               type='boolean'
               label='Grant voice to this person?'>
          <value>false</value>
        </field>
      </x>
    </options>
  </game>
</message>
```

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

Example 134. Voice Request Approval Submission

```
<message from='richardiii@shakespeare.lit/desktop'
         id='approve'
         to='england@games.shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <options>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
            <value>http://jabber.org/protocol/mmog#request</value>
        </field>
        <field var='mmog#role'>
          <value>participant</value>
        </field>
        <field var='mmog#jid'>
          <value>harritudur@shakespeare.lit/pda</value>
        </field>
        <field var='mmog#roomnick'>
          <value>earl1</value>
        </field>
        <field var='mmog#request_allow'>
          <value>true</value>
        </field>
      </x>
    </options>
  </game>
</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 MMOG 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/mmog#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".

Example 135. Admin Bans User

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='ban1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='outcast'
          jid='richardneville@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 136. Admin Bans User (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='ban1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='outcast'
          jid='richardneville@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:

Example 137. Service Informs Admin or Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='ban1'
    to='richardiii@shakespeare.lit/desktop'
    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.

Example 138. Service Removes Banned User

```
<presence
    from='england@games.shakespeare.lit'
    to='richardneville@shakespeare.lit/stabber'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='outcast' role='none'>
      <actor nick='richardneville'/>
      <reason>Treason</reason>
    </item>
    <status code='301'/>
  </game>
</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:

Example 139. Service Informs Remaining Occupants

```
<presence
    type='unavailable'
    from='england@games.shakespeare.lit'
    to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='outcast'
          jid='richardneville@shakespeare.lit/stabber'
          role='none'/>
    <status code='301'/>
  </game>
</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:

Example 140. Service Returns Error on Attempt to Ban User With Higher Affiliation

```
<iq from='england@games.shakespeare.lit'
    id='ban1'
    to='harritudur@shakespeare.lit/pda'
    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'.

Example 141. Admin Requests Ban List

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='ban2'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#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:

Example 142. Service Sends Ban List to Admin

```
<iq from='england@games.shakespeare.lit'
    id='ban2'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='outcast'
          jid='richardneville@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:

Example 143. Admin Sends Modified Ban List to Service

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='ban3'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='outcast'
          jid='richardneville@shakespeare.lit'>
      <reason>Treason</reason>
    </item>
    <item affiliation='outcast'
          jid='georgeplantagenet@shakespeare.lit'>
      <reason>Treason</reason>
    </item>
  </query>
</iq>
```

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

Example 144. Service Informs Admin of Success

```
<iq from='england@games.shakespeare.lit'
    id='ban3'
    to='richardiii@shakespeare.lit/desktop'
    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 (these matching rules are the same as those defined for privacy lists in Privacy Lists (XEP-0016)):

1. <user@domain/resource> (only that resource matches)
2. <user@domain> (any resource matches)
3. <domain/resource> (only that resource matches)
4. <domain> (the domain itself matches, as does any user@domain or domain/resource)

Some administrators might wish to ban all users associated with a specific domain from all rooms hosted by an MMOG service. Such functionality is a service-level feature and is therefore out of scope for this document; see Service Administration (XEP-0133).

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 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):

Example 145. Admin Grants Membership

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='member1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit/laptop'
          nick='sirwilliam'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 146. Admin Grants Membership (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='member1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit/laptop'
          nick='sirwilliam'>
      <reason>Good Catesby, go, effect this business soundly.</reason>
    </item>
  </query>
</iq>
```

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

Example 147. Service Informs Admin of Success

```
<iq from='england@games.shakespeare.lit'
    id='member1'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "member".

Example 148. Service Sends Notice of Membership to All Occupants

```
<presence
    from='england@games.shakespeare.lit/king'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit/laptop'
          role='participant'
          nick='sirwilliam'/>
  </game>
</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":

Example 149. Admin Revokes Membership

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='member2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='none'
          jid='earl1@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 150. Admin Revokes Membership (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='member2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='none'
          jid='earl1@shakespeare.lit'>
      <reason>Do then: but I'll not hear.</reason>
    </item>
  </query>
</iq>
```

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

Example 151. Service Informs Moderator of Success

```
<iq from='england@games.shakespeare.lit'
    id='member2'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "none".

Example 152. Service Notes Loss of Membership

```
<presence
    from='england@games.shakespeare.lit/earl1'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none'
          jid='harritudur@shakespeare.lit/pda'
          role='participant'/>
  </game>
</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.

Example 153. Service Removes Non-Member

```
<presence
    from='england@games.shakespeare.lit/earl1'
    to='richardiii@shakespeare.lit/desktop'>
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'>
      <actor nick='louisleprudent'/>
    </item>
    <status code='321'/>
  </game>
</presence>

<presence
    from='england@games.shakespeare.lit/earl1'
    to='richardiii@shakespeare.lit/desktop'>
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'/>
    <status code='321'/>
  </game>
</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 "whitelist" 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":

Example 154. Admin Requests Member List

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='member3'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='member'/>
  </query>
</iq>
```

Note: 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 the 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. 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/mmog#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.

Example 155. Service Sends Member List to Admin

```
<iq from='england@games.shakespeare.lit'
    id='member3'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit'
          nick='sirwilliam'
          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):

Example 156. Admin Sends Modified Member List to Service

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='member4'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='none'
          jid='earl1@shakespeare.lit'/>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit'/>
  </query>
</iq>
```

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

Example 157. Service Informs Moderator of Success

```
<iq from='england@games.shakespeare.lit'
    id='member4'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "none".

Example 158. Service Sends Notice of Loss of Membership to All Occupants

```
<presence
    from='england@games.shakespeare.lit/earl1'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none'
          jid='harritudur@shakespeare.lit/pda'
          role='participant'/>
  </game>
</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):

Example 159. Room Sends Invitation to New Member

```
<message
    from='england@games.shakespeare.lit'
    id='CA409450-5AAE-41C1-AAAD-5375CA738885'
    to='whastings@shakespeare.lit'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <invite from='richardiii@shakespeare.lit'/>
    <password>dieuetmondroit</password>
  </game>
</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 automatically 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:

Example 160. Service Returns Error on Attempt by Mere Member to Invite Others to a Members-Only Room

```
<message
    from='england@games.shakespeare.lit'
    id='C6E14DF6-00B7-4729-BC1C-94E59C07548E'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <invite to='thomasstanley@shakespeare.lit'>
      <reason>
        And make poor England weep in streams of blood!
      </reason>
    </invite>
  </game>
  <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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "member".

Example 161. Service Sends Notice of Membership to All Occupants

```
<presence
    from='england@games.shakespeare.lit/hastings'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='whastings@shakespeare.lit/smartphone'
          role='participant'/>
  </game>
</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":

Example 162. Admin Grants Moderator Status

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='mod1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='sirwilliam'
          role='moderator'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 163. Admin Grants Moderator Status (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='mod1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='sirwilliam'
          role='moderator'>
      <reason>Look to the drawbridge there!</reason>
    </item>
  </query>
</iq>
```

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

Example 164. Service Informs Admin of Success

```
<iq from='england@games.shakespeare.lit'
    id='mod1'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "moderator".

Example 165. Service Sends Notice of Moderator Status to All Occupants

```
<presence
    from='england@games.shakespeare.lit/king'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit/laptop'
          role='moderator'/>
  </game>
</presence>

[ ... ]
```

9.7. Revoking Moderator Status

An admin might want to revoke a user's moderator status. An admin MAY 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":

Example 166. Admin Revokes Moderator Status

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='mod2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='buckingham'
          role='participant'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 167. Admin Revokes Moderator Status (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='mod2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='buckingham'
          role='participant'>
      <reason>Will not King Richard let me speak with him?</reason>
    </item>
  </query>
</iq>
```

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

Example 168. Service Informs Admin of Success

```
<iq from='england@games.shakespeare.lit'
    id='mod2'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'role' attribute set to a value of "participant".

Example 169. Service Notes Loss of Moderator Status

```
<presence
    from='england@games.shakespeare.lit/king'
    to='harritudur@shakespeare.lit/pda'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='buckingham@shakespeare.lit/notebook'
          role='participant'/>
  </game>
</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:

Example 170. Service Returns Error on Attempt to Revoke Moderator Status from an Admin or Owner

```
<iq from='england@games.shakespeare.lit'
    id='modtest'
    to='richardiii@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'.

Example 171. Admin Requests Moderator List

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='mod3'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#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:

Example 172. Service Sends Moderator List to Admin

```
<iq from='england@games.shakespeare.lit'
    id='mod3'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit/laptop'
          nick='sirwilliam'
          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):

Example 173. Admin Sends Modified Moderator List to Service

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='mod4'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item nick='sirwilliam'
          role='moderator'/>
    <item nick='hastings'
          role='moderator'/>
  </query>
</iq>
```

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

Example 174. Service Informs Admin of Success

```
<iq from='england@games.shakespeare.lit'
    id='mod4'
    to='richardiii@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:

Example 175. Service Returns Error on Attempt to Revoke Moderator Status from an Admin or Owner

```
<iq from='england@games.shakespeare.lit'
    id='modtest'
    to='harritudur@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.

Example 176. Registration Request Approval Form

```
<message from='england@games.shakespeare.lit'
         id='407665A9-E54E-4AD5-905F-9FD8864489B3'
         to='wcatesby@shakespeare.lit/laptop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <options>
      <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/mmog#register</value>
        </field>
        <field var='mmog#register_first'
               type='text-single'
               label='Given Name'>
          <value>William</value>
        </field>
        <field var='mmog#register_last'
               type="text-single"
               label="Family Name">
          <value>Catesby</value>
        </field>
        <field var='mmog#register_roomnick'
               type="text-single"
               label="Desired Nickname">
          <value>sirwilliam</value>
        </field>
        <field var='mmog#register_url'
               type="text-single"
               label="User URL">
          <value>http://thewarsoftheroses.net/william-catesby/</value>
        </field>
        <field var='mmog#register_email'
               type="text-single"
               label="Email Address">
          <value>william.catesby@thewarsoftheroses.net</value>
        </field>
        <field var='mmog#register_faqentry'
               type="text-multi"
               label="FAQ Entry">
          <value>March on, march on, since we are up in arms.</value>
        </field>
        <field var='mmog#register_allow'
               type='boolean'
               label='Allow this person to register with the room?'>
          <value>0</value>
        </field>
      </x>
    </options>
  </game>
</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) as is done in Publish-Subscribe (XEP-0060)) 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)) are used for room configuration, triggered by use of the 'http://jabber.org/protocol/mmog' namespace. If an entity does not include the MMOG namespace in its room join/create request, then the service shall create the room and not wait for configuration via Data Forms before creating the room (this ensures backwards-compatibility with the old groupchat 1.0 protocol); however, if the room join/create request includes the MMOG extension, then 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:

Example 177. Service Informs User of Inability to Create a Room

```
<presence
    from='england@games.shakespeare.lit/earl1'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <game xmlns='http://jabber.org/protocol/mmog'
        var='http://jabber.org/protocol/mmog/dga'/>
  <error by='england@games.shakespeare.lit' type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

If a game element or the 'var' attribute with the game namespace is missing then the service MUST deny creating a room and send a presence with a bad request error back to the user.

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 Gaming protocol by including extended presence information in an empty <game/> child element qualified by the 'http://jabber.org/protocol/mmog' 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/mmog#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/mmog#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 <game/> element qualified by the 'http://jabber.org/protocol/mmog' namespace (this is the same stanza sent when seeking to enter a room).

Example 178. User Creates a Room and Signals Support for Multi-User Gaming

```
<presence
    from='richardiii@shakespeare.lit/desktop'
    to='england@games.shakespeare.lit/king'>
  <game xmlns='http://jabber.org/protocol/mmog'
        var='http://jabber.org/protocol/mmog/dga'/>
</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:

Example 179. Service Acknowledges Room Creation

```
<presence
    from='england@games.shakespeare.lit/king'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='owner'
          role='moderator'/>
    <status code='110'/>
    <status code='201'/>
  </game>
</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.

Note: If the presence stanza sent to a nonexistent room does not include an <game/> element qualified by the 'http://jabber.org/protocol/mmog' namespace as shown above, the service SHOULD create a default room without delay (i.e., it MUST assume that the client supports groupchat 1.0 rather than MUC and therefore it MUST NOT lock the room while waiting for the room creator to either accept an instant room or configure a reserved room).

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/mmog#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":

Example 180. Owner Requests Instant Room

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='create1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <options>
      <x xmlns='jabber:x:data' type='submit'/>
    </options>
  </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/mmog#owner' namespace:

Example 181. Owner Requests Configuration Form

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='create1'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <options/>
  </query>
</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.)

Example 182. Service Sends Configuration Form

```
<iq from='england@games.shakespeare.lit'
    id='create1'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <options>
      <x xmlns='jabber:x:data' type='form'>
        <title>Configuration for "england" Room</title>
        <instructions>
            Your room england@games.shakespeare.lit 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/mmog#roomconfig</value>
        </field>
        <field
            label='Natural-Language Room Name'
            type='text-single'
            var='mmog#roomconfig_roomname'/>
        <field
            label='Short Description of Room'
            type='text-single'
            var='mmog#roomconfig_roomdesc'/>
        <field
            label='Natural Language for Room Discussions'
            type='text-single'
            var='mmog#roomconfig_lang'/>
        <field
            label='Enable Public Logging?'
            type='boolean'
            var='mmog#roomconfig_enablelogging'>
          <value>0</value>
        </field>
        <field
            label='Allow Occupants to Change Subject?'
            type='boolean'
            var='mmog#roomconfig_changesubject'>
          <value>0</value>
        </field>
        <field
            label='Allow Occupants to Invite Others?'
            type='boolean'
            var='mmog#roomconfig_allowinvites'>
          <value>0</value>
        </field>
        <field
```

Note: The _whois configuration option specifies whether the room is non-anonymous (a value of "anyone"), semi-anonymous (a value of "moderators"), or fully anonymous (a value of "none", not shown here).

If there are no configuration options available, the service MUST return an empty query element to the room owner:

Example 183. Service Informs Owner that No Configuration is Possible

```
<iq from='england@games.shakespeare.lit'
    id='create1'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <options/>
  </query>
</iq>
```

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

Example 184. Owner Submits Configuration Form

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='create2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <options>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE'>
          <value>http://jabber.org/protocol/mmog#roomconfig</value>
        </field>
        <field var='mmog#roomconfig_roomname'>
          <value>england</value>
        </field>
        <field var='mmog#roomconfig_roomdesc'>
          <value>The Kingdom of England</value>
        </field>
        <field var='mmog#roomconfig_enablelogging'>
          <value>0</value>
        </field>
        <field var='mmog#roomconfig_changesubject'>
          <value>1</value>
        </field>
        <field var='mmog#roomconfig_allowinvites'>
          <value>0</value>
        </field>
        <field var='mmog#roomconfig_allowpm'>
          <value>anyone</value>
        </field>
        <field var='mmog#roomconfig_maxusers'>
          <value>10</value>
        </field>
        <field var='mmog#roomconfig_publicroom'>
          <value>0</value>
        </field>
        <field var='mmog#roomconfig_persistentroom'>
          <value>0</value>
        </field>
        <field var='mmog#roomconfig_moderatedroom'>
          <value>0</value>
        </field>
        <field var='mmog#roomconfig_membersonly'>
          <value>0</value>
        </field>
        <field var='mmog#roomconfig_passwordprotectedroom'>
          <value>1</value>
        </field>
        <field var='mmog#roomconfig_roomsecret'>
          <value>dieuetmondroit</value>
        </field>
        <field var='mmog#roomconfig_whois'>
          <value>moderators</value>
        </field>
        <field var='mmog#maxhistoryfetch'>
          <value>50</value>
        </field>
        <field var='mmog#roomconfig_roomadmins'>
          <value>richardiii@shakespeare.lit</value>
          <value>wcatesby@shakespeare.lit</value>
        </field>
```

In addition to the room configuration, the user MAY also supply a custom initial state for the match.

Example 185. Owner Submits Configuration Form Including a Constructed Match

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='create2'
    to='england@games.shakespeare.lit'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <options>
      <x xmlns='jabber:x:data' type='submit'>
        ...
      </x>
      <options xmlns='http://jabber.org/protocol/mmog/dga'>
        <x xmlns='jabber:x:data' type='submit'>
          ...
        </x>
      </options>
    </options>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      <x xmlns='jabber:x:data' type='submit'>
        ...
      </x>
    </state>
  </query>
</iq>
```

Valid states are defined by the game protocol and may consist of the explicit current state in form of a data form or the series of turns that led to the state.

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

Example 186. Service Informs New Room Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='create2'
    to='richardiii@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.

Example 187. Service Informs Owner that Requested Configuration Options Are Unacceptable

```
<iq from='england@games.shakespeare.lit'
    id='create2'
    to='richardiii@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:

Example 188. Owner Cancels Initial Configuration

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='create2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#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/mmog#owner' namespace.

Example 189. Owner Requests Configuration Form

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='config1'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#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:

Example 190. Service Denies Configuration Access to Non-Owner

```
<iq from='england@games.shakespeare.lit'
    id='configures'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <query xmlns='http://jabber.org/protocol/mmog#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:

Example 191. Service Sends Configuration Form to Owner

```
<iq from='england@games.shakespeare.lit'
    id='config1'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <options>
      <x xmlns='jabber:x:data' type='form'>
        <title>Configuration for "england" 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/mmog#roomconfig</value>
        </field>
        <field
            label='Natural-Language Room Name'
            type='text-single'
            var='mmog#roomconfig_roomname'>
          <value>england</value>
        </field>
        <field
            label='Short Description of Room'
            type='text-single'
            var='mmog#roomconfig_roomdesc'>
          <value>The Kingdom of England</value>
        </field>
        <field
            label='Enable Public Logging?'
            type='boolean'
            var='mmog#roomconfig_enablelogging'>
          <value>0</value>
        </field>
        <field
            label='Allow Occupants to Change Subject?'
            type='boolean'
            var='mmog#roomconfig_changesubject'>
          <value>0</value>
        </field>
        <field
            label='Allow Occupants to Invite Others?'
            type='boolean'
            var='mmog#roomconfig_allowinvites'>
          <value>0</value>
        </field>
        <field
            label='Who Can Send Private Messages?'
            type='list-single'
            var='mmog#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'>
```

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:

Example 192. Owner Cancels Subsequent Configuration

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='config2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#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:

Example 193. Service Notes Loss of Admin Affiliation

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='member'
          jid='wcatesby@shakespeare.lit/laptop'
          role='participant'/>
  </game>
</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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog#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":

Example 194. Service Notes Gain of Admin Affiliation to All Users

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit/laptop'
          role='moderator'/>
  </game>
</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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#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).

Example 195. Service Notes Loss of Owner Affiliation

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit/laptop'
          role='moderator'/>
  </game>
</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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog#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).

Example 196. Service Notes Gain of Owner Affiliation to All Users

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='owner'
          jid='wcatesby@shakespeare.lit/laptop'
          role='moderator'/>
  </game>
</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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace, which shall contain only a <status/> element with an appropriate value for the 'code' attribute. Here is an example:

Example 197. Configuration Status Code

```
<message from='england@games.shakespeare.lit'
         id='80349046-F26A-44F3-A7A6-54825064DD9E'
         to='richardiii@shakespeare.lit/desktop'
         type='groupchat'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <configuration-changed/>
    <status code='170'/>
    <state xmlns='http://jabber.org/protocol/mmog/dga'>
      ...
    </state>
  </game>
</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":

Example 198. Owner Grants Owner Status

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='owner1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='owner'
          jid='wcatesby@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 199. Owner Grants Owner Status (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='owner1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='owner'
          jid='wcatesby@shakespeare.lit'>
      <reason>Look to the drawbridge there!</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:

Example 200. Service Informs Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='owner1'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#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).

Example 201. Service Sends Notice of Owner Status to All Occupants

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='owner'
          jid='wcatesby@shakespeare.lit'
          role='moderator'/>
  </game>
</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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "owner".

Example 202. Service Sends Notice of Owner Status to All Occupants

```
<message
    from='england@games.shakespeare.lit'
    id='22B0F570-526A-4F22-BDE3-52EC3BB18371'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='owner'
          jid='wcatesby@shakespeare.lit'
          role='moderator'/>
  </game>
</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":

Example 203. Owner Revokes Owner Status

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='owner2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 204. Owner Revokes Owner Status (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='owner2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit'>
      <reason>A royal battle might be won and lost.</reason>
    </item>
  </query>
</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.

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:

Example 205. Service Informs Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='owner2'
    to='richardiii@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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog#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:

Example 206. Service Notes Loss of Owner Affiliation

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit'
          role='moderator'/>
  </game>
</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'.

Example 207. Owner Requests Owner List

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='owner3'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#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.

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:

Example 208. Service Sends Owner List to Owner

```
<iq from='england@games.shakespeare.lit'
    id='owner3'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='owner'
          jid='richardiii@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; *(N.B.)* each item MUST include the 'affiliation' and 'jid' attributes 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.

Example 209. Owner Sends Modified Owner List to Service

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='owner4'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='owner'
          jid='richardiii@shakespeare.lit'/>
  </query>
</iq>
```

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

Example 210. Service Returns Error on Attempt by Non-Owner to Modify Owner List

```
<iq from='england@games.shakespeare.lit'
    id='ownertest'
    to='harritudur@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:

Example 211. Service Informs Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='owner4'
    to='richardiii@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":

Example 212. Owner Grants Admin Privileges

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='admin1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 213. Owner Grants Admin Privileges (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='admin1'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit'>
      <reason>Look to the drawbridge there!</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:

Example 214. Service Informs Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='admin1'
    to='richardiii@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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#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").

Example 215. Service Sends Notice of Admin Status to All Occupants

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit'
          role='moderator'/>
  </game>
</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 <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "admin".

Example 216. Service Sends Notice of Admin Status to All Occupants

```
<message
    from='games.shakespeare.lit'
    id='C75B919A-30B3-4233-AE89-6E9834E26929'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit'
          role='none'/>
  </game>
</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).

Example 217. Owner Revokes Admin Status

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='admin2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='none'
          jid='buckingham@shakespeare.lit'/>
  </query>
</iq>
```

The <reason/> element is OPTIONAL.

Example 218. Owner Revokes Admin Status (With a Reason)

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='admin2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='none'
          jid='buckingham@shakespeare.lit'>
      <reason>Will not King Richard let me speak with him?</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:

Example 219. Service Informs Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='admin2'
    to='richardiii@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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog#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").

Example 220. Service Notes Loss of Admin Affiliation

```
<presence
    from='england@games.shakespeare.lit/buckingham'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none'
          jid='buckingham@shakespeare.lit'
          role='participant'/>
  </game>
</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 a <game/> element qualified by the 'http://jabber.org/protocol/mmog#user' namespace and containing an <item/> child with the 'affiliation' attribute set to a value other than "admin".

Example 221. Service Notes Loss of Admin Affiliation

```
<message
    from='england@games.shakespeare.lit'
    id='2CF9013B-E8A8-42A1-9633-85AD7CA12F40'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none'
          jid='buckingham@shakespeare.lit'
          role='participant'/>
  </game>
</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'.

Example 222. Owner Requests Admin List

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='admin3'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#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.

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:

Example 223. Service Sends Admin List to Owner

```
<iq from='england@games.shakespeare.lit'
    id='admin3'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='admin'
          jid='wcatesby@shakespeare.lit'
          nick='sirwilliam'/>
    <item affiliation='admin'
          jid='whastings@shakespeare.lit'/>
          nick='hastings'/>
    <item affiliation='admin'
          jid='buckingham@shakespeare.lit'/>
          nick='buckingham'/>
  </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; *(N.B.)* 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.

Example 224. Owner Sends Modified Admin List to Service

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='admin4'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#admin'>
    <item affiliation='none'
          jid='buckingham@shakespeare.lit'>
    </item>
    <item affiliation='none'
          jid='whastings@shakespeare.lit'>
    </item>
  </query>
</iq>
```

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

Example 225. Service Returns Error on Attempt by Non-Owner to Modify Admin List

```
<iq from='england@games.shakespeare.lit'
    id='admintest'
    to='harritudur@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:

Example 226. Service Informs Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='admin4'
    to='richardiii@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. Room Saving

The owner may save the room whenever the match status is 'inactive' and save the room including the match state in a moderated room.

Example 227. Owner Saves the Room

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='save'
    to='england@games.shakespeare.lit'>
    type='result'>
  <save xmlns='http://jabber.org/protocol/mmog#owner'/>
</iq>
```

If saving is allowed, the service MUST inform all occupants and remove them from the room.

Example 228. Service Broadcasts Presence to all Occupants

```
<presence
    from='england@games.shakespeare.lit/king'
    to='richardiii@shakespeare.lit/desktop'
    type='unavaiable'>
  <saved xmlns='http://jabber.org/protocol/mmog'/>
</presence>

<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='wcatesby@shakespeare.lit/laptop'
    type='unavaiable'>
  <saved xmlns='http://jabber.org/protocol/mmog'/>
</presence>
```

Example 229. Service Informs Owner of Successful Save Request

```
<iq from='england@games.shakespeare.lit'
    id='save'
    to='richardiii@shakespeare.lit/desktop'
    type='result'/>
```

After saving the room, nobody can join it until it is loaded by the owner.

10.10. Room Loading

For Discovering of Saved Rooms see the Search for Rooms secion. Send an 'iq' stanza to request loading a room as follows:

Example 230. Owner Requests Loading an Adjourned Room

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='load1'
    to='england@games.shakespeare.lit'
    type='get'>
  <load xmlns='http://jabber.org/protocol/mmog#owner'/>
</iq>
```

After loading, the match status is 'paused' if the match was 'active' before saving. Alternatively, the match status is set to 'inactive' if the match was inactive. The service SHOULD send an invitation to all occupants that were present in the game when saved.

Players entering the room SHOULD be assigned the role they had when the room was saved.

10.11. Modifying the Member List

In the context of a members-only match, the member list is essentially a "whitelist" of people who are allowed to enter the match. Anyone who is not a member is effectively banned from entering the match.

In the context of an open match, the member list is simply a list of users (bare JID and reserved nick) who are registered with the match. Such users may appear in a match roster, have their match nickname reserved, be returned in search results, and the like.

It is RECOMMENDED that only the room owner has the privilege to modify the member list in members-only rooms. To do so, the owner first requests the member list by querying the room for all users with an affiliation of "member":

Example 231. Owner Requests Member List

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='member3'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <item affiliation='member'/>
  </query>
</iq>
```

Note: 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 the room requests the member list. This functionality may 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. 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 owner qualified by the 'http://jabber.org/protocol/mmog#owner' namespace; each item MUST include the 'affiliation' and 'jid' attributes and MAY include the 'nick' and 'role' attributes for each members that is currently an occupant.

Example 232. Service Sends Member List to Owner

```
<iq from='england@games.shakespeare.lit'
    id='member3'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <item affiliation='member'
        jid='wcatesby@shakespeare.lit'
        nick='sirwilliam'/>
  </query>
</iq>
```

The owner MAY then modify the member list. In order to do so, the owner 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 and MUST NOT include the 'role' attribute (which is used to manage game roles in a room):

Example 233. Owner Sends Modified Member List to Service

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='member4'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <item affiliation='none'
        jid='earl1@shakespeare.lit'/>
    <item affiliation='none'
        jid='buckingham@shakespeare.lit'/>
  </query>
</iq>
```

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

Example 234. Service Informs Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='member4'
    to='richardiii@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 affiliation to "none" and send appropriate presence to the removed member as previously described. No matter whether the removed member was in or out of a members-only room, 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 <game/> element qualified by the 'http://jabber.org/protocol/mmog' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "none".

Example 235. Service Sends Notice of Loss of Membership to All Occupants

```
<presence
    from='england@games.shakespeare.lit/king'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <item affiliation='none'/>
  </game>
</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, for example as an owner (such a user would by definition not be in the match; note also that this example includes a password but not a reason — both child elements are OPTIONAL):

Example 236. Room Sends Invitation to New Member

```
<message
    from='england@games.shakespeare.lit'
    to='whastings@shakespeare.lit'>
  <invited xmlns='http://jabber.org/protocol/mmog#user'
      from='richardiii@shakespeare.lit'
      var='http://jabber.org/protocol/mmog/dga'>
    <password>dieuetmondroit</password>
  </invited>
</message>
```

While only the owner 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 match. In such a situation, any invitation sent SHOULD automatically trigger the addition of the invitee to the member list. However, if invitation privileges are restricted to the owner 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:

Example 237. Service Returns Error on Attempt by Mere Member to Invite Others to a Members-Only Match

```
<message
    from='england@games.shakespeare.lit'
    to='harritudur@shakespeare.lit/pda'
    type='error'>
  <invite xmlns='http://jabber.org/protocol/mmog#user'
      to='thomasstanley@shakespeare.lit'>
    <reason>
      Thou offspring of the House of Lancaster!
    </reason>
  </invite>
  <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 <game/> element qualified by the 'http://jabber.org/protocol/mmog' namespace and containing an <item/> child with the 'affiliation' attribute set to a value of "member".

Example 238. Service Sends Notice of Membership to All Occupants

```
<presence
    from='england@games.shakespeare.lit/sirwilliam'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <item affiliation='member'/>
  </game>
</presence>

...
```

10.12. Revoke and Assign Teams

The room owner may decide to modify the assigned game teams in a room. To do so, the owner first requests a list of the occupants and assigned teams by querying the room:

Example 239. Owner Requests List of All Occupants and Assigned Teams

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='roles1'
    to='england@games.shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <item team='all'/>
  </query>
</iq>
```

Note: The team 'all' is a reserved name for querying the list of active players and MUST NOT be redefined by games for other purposes.

The service SHOULD then return the full list of all occupants qualified by the 'http://jabber.org/protocol/mmog#owner' namespace; each item MUST include the 'team' and 'nick' attributes and MAY include the 'jid' and 'affiliation' attributes for each occupant in the room.

Example 240. Service Sends List of Occupants and Assigned Teams to Owner

```
<iq from='england@games.shakespeare.lit'
    id='roles1'
    to='richardiii@shakespeare.lit/desktop'
    type='result'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <item team='york' nick='king'/>
    <item team='york' nick='sirwilliam'/>
    <item team='lancaster' nick='earl1'/>
    <item team='lancaster' nick='earlofderby'/>
    <item team='none' nick='queenconsort'/>
    <item team='none' nick='hastings'/>
    <item team='none' nick='buckingham'/>
  </query>
</iq>
```

The service MUST send a <forbidden/> error if the owner has not the required Privileges.

The owner MAY then modify the roles assigned by occupants. In order to do so, the owner MUST send the changed items (i.e., only the "delta") to the service; each item MUST include the 'roles' attribute and 'nick' attribute but SHOULD NOT include the 'jid' attribute and MUST NOT include the 'affiliation' attribute:

Example 241. Owner Sends Modified List of Assigned Teams to Service

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='roles2'
    to='england@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <item team='york' nick='king'/>
    <item team='york' nick='sirwilliam'/>
    <item team='lancaster' nick='earl1'/>
    <item team='lancaster' nick='earlofderby'/>
    <item team='none' nick='queenconsort'/>
    <item team='none' nick='hastings'/>
    <item team='none' nick='buckingham'/>
  </query>
</iq>
```

The service MUST modify the roles of the occupants and then inform the owner of success:

Example 242. Service Informs Owner of Success

```
<iq from='england@games.shakespeare.lit'
    id='roles2'
    to='richardiii@shakespeare.lit/desktop'
    type='result'/>
```

The service MUST change the roles of any affected user and MUST send updated presence to all occupants, indicating the changed role by including a <game/> element qualified by the 'http://jabber.org/protocol/mmog' namespace and containing an <item/> child with the 'role' attribute.

Example 243. Service Sends Notice of Changed Roles to All Occupants

```
<presence
    from='england@games.shakespeare.lit/king'
    to='richardiii@shakespeare.lit/desktop'>
  <game xmlns='http://jabber.org/protocol/mmog'>
    <item role='york'/>
  </game>
</presence>

...
```

10.13. 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) an MMOG 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 behaviour 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/mmog#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:

Example 244. Owner Submits Room Destruction Request

```
<iq from='richardiii@shakespeare.lit/desktop'
    id='begone'
    to='wales@games.shakespeare.lit'
    type='set'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <destroy jid='wales@games.shakespeare.lit'>
      <reason>Yet to beat down these rebels here at home.</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. If extended presence information specifying the JID of an alternate location and the reason for the room destruction was provided by the room owner, the presence stanza MUST include that information.

Example 245. Service Removes Each Occupant

```
<presence
    from='wales@shakespeare.lit/king'
    to='richardiii@shakespeare.lit/desktop'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'/>
    <destroy jid='wales@games.shakespeare.lit'>
      <reason>Yet to beat down these rebels here at home.</reason>
    </destroy>
  </game>
</presence>

<presence
    from='wales@shakespeare.lit/sirwilliam'
    to='wcatesby@shakespeare.lit/laptop'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'/>
    <destroy jid='wales@games.shakespeare.lit'>
      <reason>Yet to beat down these rebels here at home.</reason>
    </destroy>
  </game>
</presence>

<presence
    from='wales@shakespeare.lit/earl1'
    to='harritudur@shakespeare.lit/pda'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'/>
    <destroy jid='wales@games.shakespeare.lit'>
      <reason>Yet to beat down these rebels here at home.</reason>
    </destroy>
  </game>
</presence>
```

Example 246. Service Informs Owner of Successful Destruction

```
<iq from='wales@games.shakespeare.lit'
    id='begone'
    to='richardiii@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:

Example 247. Service Denies Destroy Request Submitted by Non-Owner

```
<iq from='wales@games.shakespeare.lit'
    id='destroytest'
    to='wcatesby@shakespeare.lit/laptop'
    type='error'>
  <query xmlns='http://jabber.org/protocol/mmog#owner'>
    <destroy jid='wales@games.shakespeare.lit'>
      <reason>Yet to beat down these rebels here at home.</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

An MMOG 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).

If an MMOG service supports this OPTIONAL feature, it MUST include the 333 status code in the resulting presence:

Example 248. MMOG Service Removes User Because of Error

```
<presence
    from='edwardv@games.shakespeare.lit/tower'
    to='tower@shakespeare.lit/edwardv'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none' />
    <status code='110'/>
    <status code='307'/>
    <status code='333'/>
  </game>
</presence>
```

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

Example 249. MMOG Service Informs Other Occupants of Removal Because of an Error

```
<presence
    from='edwardv@games.shakespeare.lit/tower'
    to='england@shakespeare.lit/king'
    type='unavailable'>
  <game xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none'/>
    <status code='307'/>
    <status code='333'/>
  </game>
</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 MMOG service shuts downs, it SHOULD inform its participant by sending presences containing the 332 status code.

Example 250. MMOG Service Removes User Because of Service Shutdown

```
<presence
    from='edwardv@games.shakespeare.lit/tower'
    to='richardiii@shakespeare.lit/desktop'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none' />
    <status code='110'/>
    <status code='332'/>
  </x>
</presence>
<presence
    from='edwardv@games.shakespeare.lit/tower'
    to='wcatesby@shakespeare.lit/laptop'
    type='unavailable'>
  <x xmlns='http://jabber.org/protocol/mmog#user'>
    <item affiliation='none' role='none' />
    <status code='110'/>
    <status code='332'/>
  </x>
</presence>
```

12. Status Codes

Massively Multiplayer Online Gaming 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, MMOG status codes tend to follow the "philosophy" of status codes that is implicit in RFC 2616 and RFC 1893 in the following groups:

- 1xx codes are informational codes
- 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 MMOG 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.

13. Internationalization Considerations

As specified in RFC 6120, XMPP entities (including MMOG rooms and MMOG 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)).

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, MMOG 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 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, 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 MMOG, but 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 MMOG 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 MMOG 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 MMOG 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) 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) *(N.B.)*.

16. XMPP Registrar Considerations

The XMPP Registrar *(N.B.)* includes the following information in its registries.

16.1. Protocol Namespaces

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

- http://jabber.org/protocol/mmog

http://jabber.org/protocol/mmog#user

- http://jabber.org/protocol/mmog#admin
- http://jabber.org/protocol/mmog#owner

16.2. Service Discovery Category/Type

A Massively Multiplayer Online Gaming service or room is identified by the "game" category and the "multi-user" type within Service Discovery.

16.3. Service Discovery Features

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

```
<var>
  <name>http://jabber.org/protocol/mmog#register</name>
  <desc>Support for the mmog#register FORM_TYPE</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>http://jabber.org/protocol/mmog#roomconfig</name>
  <desc>Support for the mmog#roomconfig FORM_TYPE</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>http://jabber.org/protocol/mmog#roominfo</name>
  <desc>Support for the mmog#roominfo FORM_TYPE</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>http://jabber.org/protocol/mmog#stable_id</name>
  <desc>This MMOG will reflect the original message 'id' in 'groupchat' messages.</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>muc_hidden</name>
  <desc>Hidden room in Massively Multiplayer Online Gaming (MMOG)</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>muc_membersonly</name>
  <desc>Members-only room in Massively Multiplayer Online Gaming (MMOG)</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>muc_moderated</name>
  <desc>Moderated room in Massively Multiplayer Online Gaming (MMOG)</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>muc_nonanonymous</name>
  <desc>Non-anonymous room in Massively Multiplayer Online Gaming (MMOG)</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>muc_open</name>
  <desc>Open room in Massively Multiplayer Online Gaming (MMOG)</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>muc_passwordprotected</name>
  <desc>Password-protected room in Massively Multiplayer Online Gaming (MMOG)</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>muc_persistent</name>
  <desc>Persistent room in Massively Multiplayer Online Gaming (MMOG)</desc>
  <doc>XEP-xxxx</doc>
</var>
<var>
  <name>muc_public</name>
  <desc>Public room in Massively Multiplayer Online Gaming (MMOG)</desc>
  <doc>XEP-xxxx</doc>
```

16.4. Well-Known Service Discovery Nodes

The well-known Service Discovery node 'http://jabber.org/protocol/mmog#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/mmog#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) defines a process for standardizing the fields used within Data Forms qualified by a particular FORM_TYPE. Within MMOG, there are four uses of such forms:

1. Room registration (the "mmog#register" FORM_TYPE);
2. Requesting voice and approving voice requests ("mmog#request");
3. Room configuration ("mmog#roomconfig");
4. Service discovery extensions for room information ("mmog#roominfo").

The reserved fields are defined below.

16.5.1. mmog#register FORM_TYPE

Registry Submission

```
<form_type>
  <name>http://jabber.org/protocol/mmog#register</name>
  <doc>XEP-xxxx</doc>
  <desc>
    Forms enabling user registration with a
    Massively Multiplayer Online Gaming (MMOG) room or admin approval
    of user registration requests.
  </desc>
  <field
     var='mmog#register_allow'
     type='boolean'
     label='Allow this person to register with the room?'/>
  <field
      var='mmog#register_email'
      type='text-single'
      label='Email Address'/>
  <field
      var='mmog#register_faqentry'
      type='text-multi'
      label='FAQ Entry'/>
  <field
      var='mmog#register_first'
      type='text-single'
      label='Given Name'/>
  <field
      var='mmog#register_last'
      type='text-single'
      label='Family Name'/>
  <field
      var='mmog#register_roomnick'
      type='text-single'
      label='Desired Nickname'/>
  <field
      var='mmog#register_url'
      type='text-single'
      label='A Web Page'/>
</form_type>
```

16.5.2. mmog#request FORM_TYPE

Registry Submission

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

16.5.3. mmog#roomconfig FORM_TYPE