GGZ Gaming Zone Main Protocol Specification

The GGZ Gaming Zone developers

ggz-dev@mail.ggzgamingzone.org

GGZ Gaming Zone Main Protocol Specification

by The GGZ Gaming Zone developers

Main protocol specification for GGZ Gaming Zone between the GGZ server and GGZ core clients. This specification documents version 11 of the protocol.

Revision History

Revision $Revision: 8007 $ $Date: 2006-04-27 09:57:33 +0200 (jue, 27 abr 2006) $

Table of Contents

Objectives

The main GGZ protocol is the protocol which is spoken and understood by the GGZ server ggzd and core clients which are connected to it. It handles player authentication, chat and game handling. This protocol is called the GGZ Protocol, and is available in a reference implementation named libggzcore, for core client authors, written in the C programming language, and its wrappers for C++ and Python.

Chapter 1. The Protocol

The main protocol has a long history, it started out very simple and then was extended to add more and more gaming options. From version 5 on, the previous binary-opcode protocol was replaced by an extensible XML representation, which is processed with SAX parsers on both the client and the server side. Details of the protocol follow in the next chapter.

Chapter 2. Client-Server Communication Protocol

The client and GGZ server will communicate via the protocol described here. Individual games will of course communicate via their own protocol. These messages are transferred on a separate (direct) connection between game client and game server. While GGZ games are free to use whatever communication protocol they like, the GGZ protocol itself is currently encoded in XML.

Interactions are presented here categorically. For a complete reference of client-server interactions, please see the appendix.

2.1. Logging in

When a client first connects to the server, the server will start the SESSION and respond with SERVER, notifying the client of the server type, name and protocol version number. If the server is full (ie. no more user logins allowed), it will indicate this. After establishing a connection to the server, the client may send one of three commands to login:

- LOGIN is used for normal player logins. The server will respond with a RESULT tag. There are 3 types: First-time logins (which result in an account to be created and a password to be assigned), anonymous logins (which don't require a password), and registered logins.

  Note: Clients may choose to send login requests immediately, not waiting for the SERVER. This is acceptable, provided the client handle the case where the server tells that it doesn't accept any more connections (i.e. is full). It will receive no further notification that the login has failed.

After a successful login, the server may optionally send a message-of-the-day to the client via a MOTD tag. This is a text message, possibly with embedded color codes. For more details, see the part about the MOTD system in the GGZ design specification.

Any time after logging in, the client may logout of the server by closing the session, i.e. by sending the closing part of the SESSION tag. The server will end its SESSION then, too. Clients should *not* simply disconnect from the server without logging out as player data and game statistics may not get written back to the database.

Note: A client may only login once. At the present, if the player wishes to re-login for some reason (eg. to become anonymous, or to switch accounts), he will have to logout and then back in again. We may make account-switching possible at a future date, once we implement player preferences.

Warning

Although it is not explicitly disallowed by the server, clients are discouraged from allowing players to logout while at a game table. The client should send the appropriate LEAVE request first.

2.2. Requesting server information

After logging in, there are several pieces of information about the server which the client may request:

- To request a list of game types which the server supports, the client should as for the game list via the tag LIST. The server will then send the client the list of supported games via a RESULT tag which encloses a LIST. Only games which are supported by both the particular client *and* the server may be played.
- To request a list of the game rooms present on the server, clients should send a LIST tag with the room parameter. The server will then send back a LIST (again, within a RESULT tag) with the desired information.
- If the server has a message-of-the-day, it will send it to the client upon login (see the section on logging in). The client may request to see it again, however, by sending MOTD. If there is a message-of-the-day, the server will send it via MOTD. If it does not exist, the server will ignore the request.

Some infomration about the server may change while the player is logged in. If this occurs, the server will send an update notification.

2.3. Rooms

Most of the real action on a GGZ server occurs in rooms. Each room supports a particular game type, and provides a place for players to chat back and forth with friends. When a client first logs in, the player is not in a specific room, but in limbo. In order to chat with other players or join a table, the player must enter a specific room.

Changing rooms is done via ENTER. The server will respond to the room-change request with RESULT. There is no equivalent command to leave a room since leaving the room takes place automatically when a player joins some other room (We can't have players in two places at once, now can we?).

2.4. Requesting room information

Once the player has entered a room, he or she will no doubt want to know who else is there and if there are any games being played.

- To request a list of the players in the current room and what tables the are at, a client may send LIST with type 'player'. The server will respond with a list of PLAYER tags.
- Similarly, to request a list of the game tables in the room, the client should send LIST of type 'table'. The server will respond via a list of TABLE tags.

Rooms are busy places with players entering and leaving, and tables being launched and destroyed. To keep the client from having to continually resend list requests, the server will send periodic updates.

2.5. Server updates

While the player is logged in, information about the server may change, espeically if that information pertains to the players or tables in a perticular room. Rather than insist that the client send a new list request periodically, the server will send update messages to the client:

- If the the room list changes in any way, the server will send out a room UPDATE to each of the clients. Note that currently there is no mechanism to change the rooms on the fly, so this message will never occur. However, it is probable that this functionality will be added in the future.
- If another player enters or leaves the room, the server will send an UPDATE of type 'player' to all of *other* players in that room. A player will never receive a player update about himself.
- If the server's list of supported games types changes, it will send a game type UPDATE to each of the clients. Note that currently there is no mechanism to change game types on the fly, so this message will never occur. However, it is probable that this functionality will be added in the future.
- Whenever the list of tables in a room changes, each player in the room will receive an UPDATE message of type 'player'. This will occur whenever a table is launched or destroyed, a player joins or leaves, or the table's state has changed.

Note: It should be noted that these messages are asynchronous, are are sent by the server automatically without having been requested by the client.

2.6. Table Management

In order to play or watch a game with other players logged into GGZ, a player must be at a table. This is accomplished via one of two interactions:

- To join a player or spectator to an existing table, the client should send JOIN. The server will respond with RESULT, notifying the client if the attempt to join was successful.
- To launch a new table, clients must send the LAUNCH tag. The server will respond with RESULT, notifying the client of the status of the table launch.

> **Note:** Currently the server will automatically attempt to join the the player to the newly launched table. The result of this join will be sent to the client via a RESULT, even though the client did not explicitly request a join. This behaviour may change in future version of the server

At the end of a game session, the server will automatically remove all players from the table. Should a player wish to leave a table before the completion of the game, however, the client may send a LEAVE. The server will then send back RESULT. Note that not all game types will support leaving in mid-game. Such games can be left when passing the 'force' parameter, however it ends for all other players then, too.

2.7. Chatting with friends

What fun would playing games be if you can't taunt your opponents or laugh with your friends? Similar to the "Taunt" feature provided with NetSpades, the GGZ server provides the ability to send messages to other players via the server. The following interactions describe how this messaging is accomplished.

- When a player wishes to send a chat message, the client should send a CHAT. Different types of chat messages are distinguished by the chat sub-opcodes The server will respond with the status of the chat operation via CHAT
- When a player receives a chat message from another player, the server will send CHAT to the client. The chat sub-opcodes distinguish between the various chat operations

Sub-opcodes for chat messages are as follows:

- GGZ_CHAT_NORMAL signifies a "typical" chat message which is sent to all players in the current room.
- GGZ_CHAT_BEEP is a special message with no text content, which is sent to a specific player. A typical client response to the receipt of this message would be to emit a beep.
- GGZ_CHAT_PERSONAL is a private message directed to a single player.

Warning

To discourage cheating, private messages may not be sent or received while the player is at a game table.

2.8. Administrative actions

Since not everyone behaves well during chatting and playing, especially not on large anonymous networks, some help is available to privileged players for taming those who tend to be griefers.

- Players can be silenced (gagged) with or without letting them know, which is a temporary measure. They can also be kicked off a server or even banned forever. In order to initiate an administrative action against a player, an ADMIN message is to be sent by the client. It will contain the name of the affected player, probably a reason for the action, and the admin sub-opcode which identifies the action. In response, the server will return a RESULT with action being set to "admin".

Sub-opcodes for admin actions are as follows:

- GGZ_ADMIN_GAG silences the player. This works effectively like a global ignore list so that everyone but the player himself will not be bothered by his messages.
- GGZ_ADMIN_UNGAG is the inverse of GGZ_ADMIN_GAG.
- GGZ_ADMIN_KICK kicks a player from the server, i.e. closes the connection of the affected person.

2.9. Game Interactions

Since protocol version 7, direct connections have been in place. Therefore, no separate tags to embed them are needed anymore.

Appendix A. Protocol Reference

We now list a complete reference of messages which get passed to and from the GGZ server. This listing conforms to protocol version 11.

Three types of data are exchanged between the client and the server:

- char: a 1-byte signed char
- int: a 4-byte signed integer in network byte order
- string: a multibyte null-terminated string preceded by its length (including null-termination) as an integer.

Interactions take one of three forms: server messages, client requests, and server responses. Each interaction is prefaced by an opcode identifying it (some interactions consist solely of the opcode). The opcode (stored as an enumerated value) is sent as an int.

A.1. Messages sent in both directions

SESSION

Name

`SESSION` — Session start

Synopsis

```
<SESSION> ... </SESSION>
```

Description

Session start tag sent from server or client

Message Data

None

Usage

The SESSION tag is sent from the server to the client upon a successful connection. It does neither guarantee a successful login nor does it indicate if the client and server version match. It is also sent by the client, when attempting to login, or when requesting a direct connection between game client and game server.

PING

Name

`PING` — Lag measurement challenge

Synopsis

<PING ID='_id_'/>		
Data	**Type**	**Example**
ID	string	1h3k5lmfs

Description

Periodically sent request

Message Data

ID

Unique identifier for this ping; it is sent back in the pong.

Usage

The PING tag is sent periodically from the server to all clients in order to obtain their response (the PONG tag) for lag calculation. It may also be sent by the client to the server at any time to determine the client's own lag time.

PONG

Name

`PONG` — Lag measurement response

Synopsis

<PONG/>		
Data	**Type**	**Example**
ID	string	1h3k5lmfs

Description

Response to lag measurement challenge

Message Data

ID

Unique identifier for this pong - the same string that was sent in the ping.

Usage

The PONG should be sent immediately in response to the PING tag. The resulting round-trip time can be measured to determine the player's lag. When the server sends the ping, the lag measurement is tracked and reported to all players in the room (including that player). But for a lagging client - or one that is not in a room - this is not helpful. Thus the client can also send a ping to determine its own latency.

A.2. Server to client messages

ROOM

Name

ROOM — List entry describing one room

Synopsis

<ROOM ID="id" NAME="name" GAME="game" PLAYERS="players"> <DESC>desc</DESC> </ROOM>

Data	Type	Example
ID	int	0
NAME	string	TicTacToe
GAME	int	4
PLAYERS	int	14
DESC	string	Sample TTT room

Description

A list entry describing one room at a GGZ server

Message Data

ID

Unique identifier for this room

NAME

Room name

GAME

The identifier of the game associated with this room

PLAYERS

The number of players currently in the room. This value is sent in the room list and may be periodically updated. However this information is not updated immediately, so the client's knowledge may become stale. For the current room the client should use the list of players in the room as a more accurate value.

DESC

Verbose room description (sent only if requested)

Usage

A list of room tags is sent when requesting the list of available rooms on a server, typically done after logging in.

GAME

Name

GAME — List entry describing one game type

Synopsis

<GAME ID="id" NAME="name" VERSION="version"> <PROTOCOL ENGINE="engine" VERSION="version"/> <ALLOW PLAYERS="players" BOTS="bots" SPECTATORS="spectator"/> <ABOUT AUTHOR="author" URL="url"/> <DESC>desc</DESC> </GAME>

Data	Type	Example
ID	int	0
NAME	string	TicTacToe
VERSION	string	0.2
ENGINE	string	TicTacToe
PLAYERS	string	4
BOTS	string	1..3
SPECTATORS	string	true
AUTHOR	string	Anonymous Coward
URL	string	http://www.ggzgamingzone.org/gameclients/
DESC	string	My first game

Description

A list entry describing one game at a GGZ server, which may be used in one or more rooms.

Message Data

ID

Unique identifier for this game

NAME

The name of the game

VERSION

Game server program version, and game server/client communication protocol version

ENGINE

Generalized game type, which is used for frontend selection

PLAYERS

The number of players allowed in this game, in the form of a list of individual numbers and/or a single range of numbers, separated by spaces. For instance ’1 2 3 5..10’ includes both a list and a range.

BOTS

The number of AI bots allowed at a table of this game. The form is that of a number list (the same as for the PLAYERS element). Zero bots is always allowed and will not be listed.

SPECTATORS

Whether spectators are allowed or not, either ’true’ or ’false’.

AUTHOR

Name of the author or author team

URL

Pointer to the project homepage of the game server

DESC

Description for the game type offered by this server

Usage

After the client requested the list of games, a LIST containing some game entries is sent from the server to the client.

SERVER

Name

`SERVER` — Server identification

Synopsis

<SERVER ID="id" NAME="name" VERSION="version" STATUS="status"> <OPTIONS CHATLEN="chatlen"/> </SERVER>		
Data	**Type**	**Example**
ID	string	GGZ-0.0.6
NAME	string	Harry's GGZ server
VERSION	int	6
STATUS	string	ok
CHATLEN	int	512

Description

An identification message from the server

Message Data

ID

Server identification string (including version number)

VERSION

Integer version number

NAME

Descriptive name of that server

STATUS

Current server status, can be either 'ok' or 'full'

CHATLEN

Maximum length of chat messages

Usage

The SERVER tag is sent from the server to the client right after the SESSION tag. It must be examined by the client to determine whether a login is possible or not.

MOTD

Name

`MOTD` — Server Message of the day

Synopsis

<MOTD PRIORITY="priority" URL="url"> <![CDATA[...]]> </MOTD>

Data	**Type**	**Example**
PRIORITY	string	normal
URL	string	http://...

Description

Message of the day (MOTD) from the Server

Message Data

PRIORITY

MOTD priority used to determine whether to display it or not

URL

Web page containing a HTML MOTD, or empty to display text MOTD

Data (CDATA)

All lines of the MOTD, separated by the newline character

Usage

The MOTD tag is sent from the server to the client upon successful login (ie. after the RESULT tag belonging to a LOGIN request)

UPDATE

Name

`UPDATE` — Notification that the list of players or the list of tables in a room has changed, or the list of rooms or game types on a server.

Synopsis

<UPDATE TYPE="type" ACTION="action" ROOM="room" FROMROOM="from" TOROOM="to">		
Data	**Type**	**Example**
TYPE	string	player
ACTION	string	add
ROOM	int	1
FROMROOM	int	2
TOROOM	int	3

Description

Notification from the server that the list of players in the current room has been modified since the last update, or that a table has been added, removed or changed its state, or that a room or game type has been added or removed.

Message Data

TYPE

Update type. Can be ’player’ or ’table’ for room updates, or ’room’ or ’game’ for server updates.

ACTION

Action to do. Can be ’add’, ’delete’, ’lag’, ’perms’, or ’stats’ for players; ’add’, ’delete’, ’join’,

’leave’, ’status’, ’desc’, or ’seat’ for tables; ’add, ’delete’, ’close’, or ’players’ for rooms. No game updates are possible at this time.

ROOM

Room to which the update applies. Only used for types ’player’ and ’table’.

FROMROOM

Room the player is coming from. Only used for type ’players’ with action ’add’. The room may be -1 indicating "no room".

TOROOM

Room the player is going to. Only used for type ’players’ with action ’delete’. The room may be -1 indicating "no room".

Usage

An UPDATE tag is sent from the server to the client if the list of players, tables, rooms or games has changed since the last update. It is *not* sent upon initial entry to a room or server; the information is originally sent in a LIST. An update is sent when the information of that type changes.

The UPDATE tag contains one or more tags of type PLAYER, TABLE, ROOM, or GAME. The information included in the subtags depends on the ACTION attribute of the update. For instance a "table" "desc" update would include only the description of the table, while a "player" "add" update includes the full information about the player. A full list of these dependencies is not available at this time.

PLAYER

Name

`PLAYER` — List entry describing one player.

Synopsis

<PLAYER ID=’id’ TYPE=’type’ TABLE=’table’ PERMS=’perms’ LAG=’lag’ WINS=’wins’ LOSSES=’losses’ TIES=’ties’ FORFEITS=’forfeits’ RATING=’rating’ RANKING=’rating’ HIGHSCORE=’highscore’/>

Data	Type	Example

ID	string	Grubby
TYPE	string	guest
TABLE	int	-1
PERMS	int/hex	0x0000000F
LAG	int	1
WINS	int	7
LOSSES	int	3
TIES	int	1
RATING	int	1500
RANKING	int	3
HIGHSCORE	long	49807

Description

List entry containing the description of one player

Message Data

ID

Unique identifier of this player

TYPE

Player type (can be 'guest', 'normal', 'admin' or 'bot')

TABLE

ID number of table at which player is "sitting"; -1 or not present if the player is not at a table.

PERMS

Player permission set - an integer bitfield giving which permissions the player has, according to the enumeration in ggz_common.h.

LAG

Lag value of the player, which ranges from zero (ideal connection) to five (slow connection).

WINS

The number of wins the player has. This will be a non-negative integer, if present. It will not be present if the player has no stats of this type (for whatever reason). WINS, LOSSES, TIES, and FORFEITS will all be either present or not present (as a group).

LOSSES

The number of losses the player has - a non-negative integer. See WINS.

TIES

The number of ties the player has - a non-negative integer. See WINS.

FORFEITS

The number of forfeits (abandoned games) the player has - a non-negative integer. See WINS.

RATING

If present, it represents the player's rating. This is generally done on an ELO scale, although the parameters may differ from those used for Chess. It will not be given if the player has no rating.

RANKING

If present, provides the absolute ranking of the player. This will be a positive integer. It will not be given if the player has no ranking.

HIGHSCORE

If present, provides the highest score the player has achieved at the current game. It will not be given if there is no available highscore for this player.

Usage

The PLAYER is contained in a list sent from the server to the client in response to LIST request tag of type 'player'. It may also be contained within the UPDATE tag send from the server when player information changes. If sent as part of a LIST then the information will be complete - and any omissions are intentional. If sent as part of an UPDATE then only some attributes will be present.

TABLE

Name

`TABLE` — List entry containing one table

Synopsis

<TABLE ID="id" GAME="game" STATUS="status" SEATS="seats"> <DESC>desc</DESC> <SEAT NUM="num" TYPE="type">player</SEAT> ... </TABLE>		
Data	**Type**	**Example**

ID	int	0
GAME	int	0
STATUS	int	1
SEATS	int	2
DESC	string	An empty table
NUM	int	0
TYPE	string	player
PLAYER	string	Grubby

Description

The TABLE tag describes one table with all its properties

Message Data

ID

Unique identifier of this table in this room

GAME

Associated game. Only used in updates with action ’add’.

STATUS

Current status of the table. Only used within updates with action ’status’ or ’add’.

SEATS

Number of seats on this table.

DESC

Description of the table. Only sent when table is being added to the list, i.e. in updates with action ’add’.

NUM

Number of a seat.

TYPE

Type of a seat, which can be ’open’, ’player’, ’reserved’ or ’bot’.

PLAYER

Player name in case the type is either ’player’ or ’reserved’. It is also included if a ’bot’ seat is reserved for a named bot.

Usage

TABLE tags are sent in a list from the server to the client in response to LIST when LIST is of type 'table', and in UPDATE tags of type 'table'.

JOIN

Name

`JOIN` — Message to indicate you've joined a table

Synopsis

<JOIN TABLE='table' SPECTATOR='spectator'/>		
TABLE	integer	1
SPECTATOR	boolstring	false

Description

Tells the client that they have joined a table. Currently each client and each player can only be at one table at a time. The join will be initiated when the client sends either a JOIN request to join an existing table or a LAUNCH request to launch a new table (and subsequently join it).

Message Data

TABLE

The index of the table that has been joined. This will be a non-negative integer corresponding to the index sent in the TABLE tag.

SPECTATOR

Either "true" if the player is a spectator or "false" if they are a normal player. This is generally determined based upon the client's JOIN request.

Usage

JOIN is sent from the server to the client to inform the player they've joined a table. The client should take any necessary actions.

LEAVE

Name

LEAVE — Message to indicate you've left a table

Synopsis

<LEAVE REASON='reason' PLAYER='player'/>		
REASON	string	normal
PLAYER	player	jdorje

Description

Tells the client that they have left a table. This may be initiated by the player sending the server a LEAVE request, by the game server exiting when the game is over, by the player being booted from the table, or by the game server aborting. In any of the cases the client should make sure the game client exits normally.

Message Data

REASON

The reason the player left the table, currently one of "normal", "boot", "gameover", or "gameerror".

PLAYER

The name of the (remote) player responsible for the leave. This is currently only provided for the "boot" leave, where it gives the name of the player who initiated the boot.

Usage

LEAVE is sent from the server to the client to tell of a table leave. The client should take any necessary actions.

RESULT

Name

`RESULT` — General message to indicate the result of a request

Synopsis

<RESULT ACTION="action" CODE="code"/>

Data	Type	Example
ACTION	string	list
CODE	string	ok

Description

Server response to any request, embedding the answer data

Message Data

ACTION

The request type this result is referring to. Actions include 'motd', 'list', 'enter', 'chat', 'launch', 'join', 'leave', 'reseat', 'update', 'protocol', 'channel', 'login', and 'pong'.

CODE

Result indicator: either 'ok' or an error string. Errors may include 'usr lookup', 'bad options', 'room full', 'table full', table empty', launch fail', 'join fail', 'no table', 'leave fail', 'leave forbidden', 'already logged in', 'not logged in', 'not in room', 'at table', 'in transit', 'no permission', 'bad xml', 'seat assign fail', 'no channel', or 'too long'. The client should behave sanely when it receives an unrecognized error.

Usage

RESULT is sent from the server to the client in response to LIST, ENTER, LAUNCH, JOIN, LEAVE, CHAT, ADMIN, PERMADMIN, and LOGIN requests. Note that in some of these cases (e.g. LIST) the response to the request will be contained within the RESULT, whereas in others it will not. The logic on which is which is reasonable, but nonetheless the lack of consistency may be confusing.

LIST

Name

`LIST` — Server response to request for list of rooms or list of games

Synopsis

<LIST TYPE="type" ROOM="room"> </LIST>		
Data	**Type**	**Example**
TYPE	string	room
ROOM	int	1

Description

Server response to request for list of rooms or game types

Message Data

TYPE

Type of list, either 'room', 'game', 'table' or 'player'

ROOM

The room to which this list belongs. Only used for 'table' and 'player'.

Usage

The LIST is sent from the server to the client in response to the client request named LIST, and may

contain either TABLE or PLAYER or GAME or ROOM entries.

CHAT

Name

`CHAT` — Server response to chat message request

Synopsis

<CHAT TYPE="type" FROM="from"> <![CDATA[...]]> </CHAT>

Data	Type	Example
TYPE	string	private
FROM	string	Grubby

Description

Server response to player chat request

Message Data

TYPE

Type of chat message, which can be one of "normal", "announce", "beep", "private", or "table".

FROM

Player from which this message originates

Usage

The CHAT response tag is sent from the server to client to handle chat message requests via the CHAT tag.

A.3. Client to server messages

LOGIN

Name

`LOGIN` — Client requested login or registration

Synopsis

<LOGIN TYPE="type"> <NAME>name</NAME> <PASSWORD>password</PASSWORD> <EMAIL>email</EMAIL> </LOGIN>

Data	Type	Example
TYPE	string	guest
NAME	string	Gandalf
PASSWORD	string	xxxx
EMAIL	string	player@ggzcommunity.org

Description

Client requested login or registration

Message Data

TYPE

Type of login, which can be one of "normal", "guest" or "first"

NAME

Login name

PASSWORD

Player password, which is only used for registered ("normal") players, or (optional) for the registration ("first")

EMAIL

Player email address for password retrieval, only used for the registration ("first")

Usage

The LOGIN tag is sent from the client to the server to request a player login. The request may come after SERVER. The client must have started the SESSION in advance. The server will respond to the request with a RESULT tag. There are three ways to use this tag: To login as a guest player (only name needed), to login as a registered player (name and password needed), or to register for the first time (name needed, password and email optional). If no password is given for the registration, the server will assign one.

LIST

Name

`LIST` — Client request for list of rooms or games on the server, or of tables or players in a room

Synopsis

<LIST TYPE="type" FULL="full"/>

Data	Type	Example
TYPE	string	room
FULL	boolstring	true

Description

Client request for list of rooms or game types on the server, or (when in a room) for list of tables or list of players

Message Data

TYPE

The type may be 'room' for the room list, and 'game' to receive the game types list. Likewise, 'player' and 'table' are used to retrieve room information.

FULL

Verbosity flag. Set to 'false' for short room descriptions, 'true' for full descriptions. This is only used for the 'room' type.

Usage

The <LIST> tag is sent from the client to the server to request a list of rooms, game types, tables or players. The server will respond with the appropriate RESULT tag which contains a LIST. This request is only valid once the player has successfully logged in for 'room' and 'game' types (ie. after the login RESULT), and only after the player has entered a room (for 'table' and 'player' types).

LAUNCH

Name

`LAUNCH` — Client request for new table launch

Synopsis

<LAUNCH> ... </LAUNCH> **Data**	**Type**	**Example**
No data		

Description

Client request for new table launch

Usage

The LAUNCH tag request a table to be launched in the current room. Therefore, each LAUNCH request contains one tag of type TABLE. The server will respond with a RESULT tag, followed by an UPDATE of type 'table' upon success.

JOIN

Name

`JOIN` — Client request to join a table

Synopsis

<JOIN TABLE="table" SPECTATOR="spectator">		
Data	**Type**	**Example**
TABLE	int	0
SPECTATOR	boolstring	true
SEAT	int	3

Description

Client request to join table

Message Data

TABLE

The identifier of the table the player wishes to join

SPECTATOR

Whether to join as spectator (when true) or not (when false or omitted).

SEAT

The seat number of the seat to join in. If the seat number is negative or not given, the first available seat will be joined.

Usage

JOIN is sent from client to the server to request a table join. The server will respond with RESULT telling us the success of the action. If the action was successful the client will also receive a JOIN, which we should act upon. If the join succeeds it is likely an UPDATE of type 'table' will also be necessary.

LEAVE

Name

`LEAVE` — Client request to leave table

Synopsis

<LEAVE FORCE="force">		
FORCE	boolstring	false

Description

Server response to player request to leave tablet

Message Data

FORCE

Force leaving the table even though the game doesn't support it. The game would then be cancelled. For spectators, this flag is ignored.

Usage

LEAVE is sent from the client to the server to request a table leave. The server will respons with a RESULT message as well as a LEAVE message with reason "normal" if the leave succeeded.

CHAT

Name

`CHAT` — Client chat message request

Synopsis

<CHAT TYPE='type' TO='to'> <![CDATA[...]]> </CHAT>		
Data	**Type**	**Example**
TYPE	string	normal
TO	string	josef

Description

Client chat message request

Message Data

TYPE

Chat type. Can be one of "normal", "announce", "beep", "private", or "table".

TO

Name of player who will receive message. This attribute is needed for 'beep' and 'private' messages, but not for 'normal', 'announce', or 'table'. The client should behave sanely if this contains incorrect data or is not present.

Usage

A CHAT request is sent from the client to send a chat message. The server will respond with an appropriate RESULT which embeds a CHAT tag.

ADMIN

Name

`ADMIN` — Room Administration

Synopsis

```
<ADMIN ACTION='action' PLAYER='player'> <REASON><![CDATA[...]]></REASON>
</ADMIN>
```

Data	Type	Example
ACTION	string	gag
PLAYER	string	someplayer22

Description

Room administration

Message Data

ACTION

Type of administrative action. Is one of either "gag", "ungag" or "kick".

PLAYER

Name of player who is affected by the action.

REASON

A textual explanation of the action. This is mandatory for "kick" actions, and ignored for all others.

Usage

An ADMIN request is sent from the client to do an administrative action on a player in the current room. The server will respond with an appropriate RESULT which embeds a admin tag.

PERMADMIN

Name

`PERMADMIN` — Permissions administration

Synopsis

<PERMADMIN PLAYER='player' PERM='perm' VALUE='value'/>		
Data	**Type**	**Example**

PLAYER	string	someplayer22
PERM	string	join_table
VALUE	boolean	true

Description

Permissions administration

Message Data

PLAYER

Name of player who is affected by the action.

PERM

Name of the permission being modified. Current permissions include 'join_table', 'launch_table', 'rooms_login', 'rooms_admin', 'chat_announce', 'chat_bot', 'no_stats', 'edit_tables', 'edit_privmsg'.

VALUE

Whether to set or unset the permission. Either 'true' or 'false'.

Usage

Administrators can change permissions of other players. A PERMADMIN request is sent from the client to change a particular permission on a particular player. The server will respond with an appropriate RESULT which embeds an admin tag.

ENTER

Name

`ENTER` — Request to change rooms

Synopsis

<ENTER ROOM="room">		
Data	**Type**	**Example**
ROOM	int	1

Description

Request to change rooms

Message Data

ROOM

The room identifier of requested destination room

Usage

ENTER is sent from the client to the server to request a room change. It is expected that the server will respond with RESULT.

CHANNEL

Name

`CHANNEL` — Request for a direct game connection

Synopsis

<CHANNEL ID="id">		
Data	**Type**	**Example**
ID	string	Grubby

Description

Request for a direct connection between game server and game client

Message Data

ID

Identifier of the created channel. This equals the name of the player who launches or joins a game.

Usage

To obtain a channel, the CHANNEL tag is sent within a separate SESSION.