

# **GGZ Gaming Zone Server/Game Server Protocol Specification**

**The GGZ Gaming Zone developers**

**`ggz-dev@mail.ggzgamingzone.org`**

# **GGZ Gaming Zone Server/Game Server Protocol Specification**

by The GGZ Gaming Zone developers

Copyright © 2005, 2006 The GGZ Gaming Zone developers

Module protocol specification for GGZ Gaming Zone game servers. This document covers the communication between the GGZ server and the game server modules.

## Revision History

Revision \$Revision: 8007 \$ \$Date: 2006-04-27 09:57:33 +0200 (Do, 27 Apr 2006) \$

# Table of Contents

Objectives .....	iv
<b>1. The Protocol .....</b>	<b>1</b>
1.1. Startup .....	1
1.2. Pregame phase .....	1
1.3. Playing phase .....	2
1.4. Done phase .....	2
<b>A. Protocol Reference .....</b>	<b>3</b>
A.1. Messages from ggzd to game server .....	3
GAME_LAUNCH .....	3
GAME_SEAT .....	4
GAME_SPECTATOR_SEAT .....	4
GAME_RESEAT .....	5
GAME_STATE .....	6
A.2. Messages from game server to ggzd .....	7
LOG .....	7
GAME_STATE .....	8
NUM_SEATS .....	8
BOOT .....	9
BOT .....	10
OPEN .....	11
GAME_REPORT .....	12
SAVEGAME_REPORT .....	12
A.3. Symbolic identifiers and their values .....	13
ControlToTable .....	14
TableToControl .....	14
GGZSeatType .....	15
GGZTableState (GGZdModState) .....	15

# Objectives

Game servers are started by the GGZ Gaming Zone server ggzd whenever a player launches a table. The game setup is passed to the game over a special connection, as are the scores and game results from the game server to ggzd. This protocol is called the Server/Game Server Protocol, and is available in a reference implementation named libggzdmod, written in the C programming language, and its wrappers for C++ and Python.

# Chapter 1. The Protocol

Communication between server and game server happens by means of binary tokens (opcodes), which are of type integer, followed by zero or more opcode-specific variables which can be of type integer, character, or string.

At each point in time, a game server happens to be in a specific state. Messages received from the server may lead to state changes, as may some explicit transitions being executed by the game server itself. A list of all states can be found in the appendix of states.

Several actions refer to seats on the table the game is being played on. Each seat can be either empty or have an assignment. A full list can be found in the appendix of seat assignments.

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

## 1.1. Startup

The GGZ server ggzd and the game server communicate via a connection which gets established using `socketpair()`. The game server can then access the communication channel on a certain file descriptor. Usually this descriptor has the value 3, but if the environment variable `GGZSOCKET` is set, its value should be used instead. Likewise, the environment variable `GGZMODE` can be queried to see if the game server is running on GGZ at all.

The first message which will arrive is the `GAME_LAUNCH` message, used to configure the game based on the table information: how many players are going to participate, are there any reserved seats, and so on. The game server will then notify ggzd about it having the data received, and set its internal status from `CREATED` to `WAITING`.

## 1.2. Pregame phase

Now that the table is created, all the bots are going to join it automatically, but humans will join one after the other. Each of them triggers a `GAME_SEAT` message, consisting of the name, player type and, most important, the file descriptor to access the player's game client.

If the game server allows spectators to watch a game, they will be handled similarly, by triggering a `GAME_SPECTATOR_SEAT` message. Since a game might allow a spectator to become a player and vice-versa, a special `GAME_RESEAT` message might appear at any time.

The abovementioned state change is done via the `GAME_STATE` request, sent by the game server to indicate that its internal state has changed. In response to a game server's `GAME_STATE` request, ggzd will always send a `GAME_STATE` response, with no further data attached. After the `PREGAME` phase, the state will most likely be changed to `PLAYING`.

## 1.3. Playing phase

It is a good idea to report the important game events to the outside. This can be done by letting the game server send a `LOG` message to ggzd, so that it gets recorded according to the server configuration. It might end up in a log file, in a debug console or nowhere at all.

Other than that, there's not too much happening between game server and ggzd in this phase of the game, for most communication will happen between game server and game client. However, if a player leaves or another one wants to join and can't because a bot occupies the seat, the game server can request to change the seats on the table. To change the seat number, a `NUM_SEATS` request is sent. To boot a player, the `BOOT` request can be used. Finally, bots can be inserted and removed using the `BOT` and `OPEN` requests.

## 1.4. Done phase

A game is expected to change its state to `DONE` when the game is over. In most cases, a winner (or tie game) will be determined. The game results, including winner, score and the like, can be reported back to ggzd via the `GAME_REPORT` message. If a game log or savegame has been kept, the filename of it can be reported as well for further reference, using a `SAVEGAME_REPORT` message.

# Appendix A. Protocol Reference

## A.1. Messages from ggzd to game server

### GAME\_LAUNCH

#### Name

GAME\_LAUNCH — Initializes the game with its seat data

#### Synopsis

GAME_LAUNCH ...		
Data	Type	Example
Opcode	ControlToTable	GAME_LAUNCH
Game module description file name, without suffix	string	tictactoe
Number of seats at the table	integer	2
Number of spectators	integer	0
ENTRYTBL not supported.		

#### Description

Initialization of game server with table settings

#### Message Data

None

#### Usage

Sent on game startup to configure the game server according to the table configuration as performed by the client of the game host. The player name is only sent for seats of type GGZ\_SEAT\_RESERVED or GGZ\_SEAT\_BOT, and may be empty ("") for anonymous bots.

# GAME\_SEAT

## Name

GAME\_SEAT — Informs the game about seat change

## Synopsis

GAME_SEAT ...		
Data	Type	Example
Opcode	ControlToTable	GAME_SEAT
Seat number	integer	0
Seat type	GGZSeatType	GGZ_SEAT_PLAYER
Player name	string	player42
ENTRYTBL not supported.		

## Description

Informs the game about seat changes

## Message Data

None

## Usage

Whenever a player or bot player joins or leaves a table, this message is sent to the game server to notify it of the new seat assignment. The file descriptor is sent only if the seat type equals GGZ\_SEAT\_PLAYER.

# GAME\_SPECTATOR\_SEAT

## Name

GAME\_SPECTATOR\_SEAT — Informs the game about spectator seat change



## Synopsis

GAME_SPECTATOR_SEAT ...		
Data	Type	Example
Opcode	ControlToTable	GAME_SPECTATOR_SEAT
Spectator seat number	integer	0
Spectator name	string	spectator28
ENTRYTBL not supported.		

## Description

Informs the game about spectator seat change

## Message Data

None

## Usage

This message is generated whenever a spectator joins or leaves the table. The file descriptor is sent only if the spectator has joined and thus the spectator seat is occupied.

# GAME\_RESEAT

## Name

GAME\_RESEAT — Notification of seat/spectator seat change

## Synopsis

GAME_RESEAT ...		
Data	Type	Example
Opcode	ControlToTable	GAME_RESEAT
Number of old seat	int	1

Was old seat a spectator seat?	integer/boolean	1
Number of new seat	int	4
Is new seat a spectator seat?	integer/boolean	0

## Description

Notification of seat/spectator seat change

## Message Data

None

## Usage

The STAND and SIT actions allow a player to become a spectator and vice-versa. Both actions will generate this message.

# GAME\_STATE

## Name

GAME\_STATE — State update

## Synopsis

GAME_STATE ...		
Data	Type	Example
Opcode	ControlToTable	GAME_STATE

## Description

State update

**Message Data**

None

**Usage**

Acknowledgement of received GAME\_STATE request from the game server.

## A.2. Messages from game server to ggzd

### LOG

**Name**

LOG — Log message from the game

**Synopsis**

LOG ...		
Data	Type	Example
Opcode	TableToControl	LOG
Log message	string	hello world

**Description**

Log message from the game

**Message Data**

None

## Usage

The game server can write out log messages to ggzd using this message. The text will then be written or ignored according to the ggzd configuration.

# GAME\_STATE

## Name

GAME\_STATE — Request of game state change

## Synopsis

GAME_STATE ...		
Data	Type	Example
Opcode	TableToControl	GAME_STATE
Requested game state	GGZTableState (as char)	STATE_DONE

## Description

Request of game state change

## Message Data

None

## Usage

Notifies about the game server's decision to change the state to the given one. All changes will be acknowledged by the server with a GAME\_STATE response.

# NUM\_SEATS

## Name

NUM\_SEATS — Request of seat count change

## Synopsis

NUM_SEATS ...		
Data	Type	Example
Opcode	TableToControl	NUM_SEATS
Number of seats	int	12

## Description

Request of seat count change

## Message Data

None

## Usage

While the initial number of seat can range between two and the maximum number of possible seats for the gametype in use, a game server can always resize its table afterwards using this message.

# BOOT

## Name

BOOT — Request of player/spectator boot

## Synopsis

---

BOOT ...		
Data	Type	Example
Opcode	TableToControl	BOOT
Name of player to boot	string	player42

## Description

Request of player/spectator boot

## Message Data

None

## Usage

The seat in question, which can be a player or a spectator seat, is emptied by booting the player or spectator occupying it.

# BOT

## Name

BOT — Request to fill seat with bot player

## Synopsis

BOT ...		
Data	Type	Example
Opcode	TableToControl	BOT
Number of seat to use for bot	int	2

## Description

Request to fill seat with bot player

## Message Data

None

## Usage

The seat in question is occupied with a bot player, who will thus join the game.

# OPEN

## Name

OPEN — Request to open seat

## Synopsis

OPEN ...		
Data	Type	Example
Opcode	TableToControl	OPEN
Number of seat to open	int	3

## Description

Request to open seat

## Message Data

None

## Usage

A seat either occupied by a bot or reserved for a player will be opened up again so another player or bot player can occupy it.

# GAME\_REPORT

## Name

GAME\_REPORT — Notification of game results

## Synopsis

GAME_REPORT ...		
Data	Type	Example
Opcode	TableToControl	GAME_REPORT
Number of players/seats	int	5
ENTRYTBL not supported.		

## Description

Notification of game results

## Message Data

None

## Usage

When the game has ended, the game results including player positions and highscores, as well as team information, is reported back to ggzd for storage in the database.



# SAVEGAME\_REPORT

## Name

SAVEGAME\_REPORT — Notification of temporary savegame location

## Synopsis

SAVEGAME_REPORT ...		
Data	Type	Example
Opcode	TableToControl	SAVEGAME_REPORT
Name of the savegame token	string	chess2005.pgn

## Description

Notification of temporary savegame location

## Message Data

None

## Usage

A continuously game log or a final savegame can be reported to ggzd using this message. The naming is up to the game server, it might refer to a file or directory name. Note: The message GAME\_REPORT will use this value to write it into the database.

## A.3. Symbolic identifiers and their values

### ControlToTable

#### Name

`ControlToTable` — Opcodes from GGZ server to the game server module

#### Synopsis

Identifier	Value	Description
GAME_LAUNCH	0	message
GAME_SEAT	1	message
GAME_SPECTATOR_SEAT	2	message
GAME_RESEAT	3	message
GAME_STATE	4	response

#### Description

All opcodes are of type integer.

### TableToControl

#### Name

`TableToControl` — Opcodes from game server modules to the GGZ server

#### Synopsis

Identifier	Value	Description
LOG	0	message
GAME_STATE	1	request
NUM_SEATS	2	request
BOOT	3	request
BOT	4	request

OPEN	5	request
GAME_REPORT	6	message
SAVEGAME_REPORT	7	message

## Description

All opcodes are of type integer.

## GGZSeatType

### Name

GGZSeatType — Possible seat assignments for a table

## Synopsis

Identifier	Value	Description
GGZ_SEAT_NONE	0	Not initialized yet (invalid)
GGZ_SEAT_OPEN	1	Initialized to open, will be filled later
GGZ_SEAT_BOT	2	Internal or external AI player
GGZ_SEAT_PLAYER	3	Human player
GGZ_SEAT_RESERVED	4	Reserved for AI or human player of a certain name

## Description

All seat types are of type integer.

## GGZTableState (GGZdModState)

### Name

GGZTableState (GGZdModState) — Possible game states for a table

### Synopsis

Identifier	Value	Description
STATE_CREATED	0	...
STATE_WAITING	1	...
STATE_PLAYING	2	...
STATE_DONE	3	...

### Description

All states are of type integer.