GGZdMod Reference Manual

0.0.14

Generated by Doxygen 1.5.1

Fri Nov 30 14:59:07 2007

Contents

Chapter 1

GGZdMod Class Index

1.1 GGZdMod Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

GGZdMod File Index

2.1 GGZdMod File List

Here is a list of all documented files with brief descriptions:

Chapter 3

GGZdMod Class Documentation

3.1 GGZSeat Struct Reference

A seat at a GGZ game table.

`#include <ggzdmod.h>`

Public Attributes

- unsigned int **num**
- GGZSeatType **type**
- const char * **name**
- int **fd**
- void * **playerdata**

3.1.1 Detailed Description

A seat at a GGZ game table.

Each seat at the table is tracked like this.

3.1.2 Member Data Documentation

3.1.2.1 unsigned int GGZSeat::num

Seat index; 0..(num_seats-1).

3.1.2.2 GGZSeatType GGZSeat::type

Type of seat (undefined for spectators).

3.1.2.3 const char* GGZSeat::name

Name of player occupying seat.

3.1.2.4 int GGZSeat::fd

fd to communicate with seat occupant.

3.1.2.5 void* GGZSeat::playerdata

Arbitrary pointer holding seat state data

The documentation for this struct was generated from the following file:

- **ggzdmod.h**

Chapter 4

GGZdMod File Documentation

4.1 ggzdmod.h File Reference

Common functions for interfacing a game server and GGZ.

`#include <ggz.h>`

`#include <ggz_common.h>`

Classes

- struct **GGZSeat**

  A seat at a GGZ game table.

Defines

- #define **GGZDMOD_VERSION_MAJOR** 0
- #define **GGZDMOD_VERSION_MINOR** 0
- #define **GGZDMOD_VERSION_MICRO** 14
- #define **GGZDMOD_VERSION_IFACE** "6:0:0"
- #define **GGZSpectator GGZSeat**

Typedefs

- typedef **GGZdMod GGZdMod**

  A GGZdmod object, used for tracking a ggzd<->table connection.

- typedef void(∗) **GGZdModHandler** (**GGZdMod** ∗mod, **GGZdModEvent** event, const void ∗data)

  Event handler prototype.

Enumerations

- enum **GGZdModState** { **GGZDMOD_STATE_CREATED**, **GGZDMOD_STATE_WAITING**, **GGZDMOD_STATE_PLAYING**, **GGZDMOD_STATE_DONE** }

  Table states.

- enum **GGZdModEvent** {

  GGZDMOD_EVENT_STATE, **GGZDMOD_EVENT_JOIN**, **GGZDMOD_EVENT_LEAVE**, **GGZDMOD_EVENT_SEAT**,

  GGZDMOD_EVENT_SPECTATOR_JOIN, **GGZDMOD_EVENT_SPECTATOR_LEAVE**, **GGZDMOD_EVENT_SPECTATOR_SEAT**, **GGZDMOD_EVENT_PLAYER_DATA**,

  GGZDMOD_EVENT_SPECTATOR_DATA, **GGZDMOD_EVENT_ERROR** }

  Callback events.

- enum **GGZdModType** { **GGZDMOD_GGZ**, **GGZDMOD_GAME** }

  The "type" of ggzdmod.

- enum **GGZGameResult** {

  GGZ_GAME_WIN, **GGZ_GAME_LOSS**, **GGZ_GAME_TIE**, **GGZ_GAME_FORFEIT**,

  GGZ_GAME_NONE }

Functions

- int **ggzdmod_is_ggz_mode** (void)

  Is the program running in GGZ mode?

- **GGZdMod** * **ggzdmod_new** (**GGZdModType** type)

  Create a new ggzdmod object.

- void **ggzdmod_free** (**GGZdMod** *ggzdmod)

  Destroy a finished ggzdmod object.

- int **ggzdmod_get_fd** (**GGZdMod** *ggzdmod)

  Get the file descriptor for the GGZdMod socket.

- **GGZdModType** **ggzdmod_get_type** (**GGZdMod** *ggzdmod)

  Get the type of the ggzdmod object.

- **GGZdModState** **ggzdmod_get_state** (**GGZdMod** *ggzdmod)

  Get the current state of the table.

- int **ggzdmod_get_num_seats** (**GGZdMod** *ggzdmod)

  Get the total number of seats at the table.

- **GGZSeat** **ggzdmod_get_seat** (**GGZdMod** *ggzdmod, int seat)

Get all data for the specified seat.

- char * **ggzdmod_get_bot_class** (**GGZdMod** *ggzdmod, const char *name)

  Return class for named bot.

- void **ggzdmod_set_playerdata** (**GGZdMod** *ggzdmod, bool is_spectator, int seat_num, void *playerdata)

  Set a playerdata pointer.

- void * **ggzdmod_get_gamedata** (**GGZdMod** *ggzdmod)

  Return gamedata pointer.

- void **ggzdmod_set_gamedata** (**GGZdMod** *ggzdmod, void *data)

  Set gamedata pointer.

- int **ggzdmod_get_max_num_spectators** (**GGZdMod** *ggzdmod)

  Get the maximum number of spectators. This function returns the maximum number of spectator seats available. A game can use this to iterate over the spectator seats to look for spectators occupying them. Since spectators may come and go at any point and there is no theoretical limit on the number of spectators, you should consider this value to be dynamic and call this function again each time you're looking for spectators.

- **GGZSeat ggzdmod_get_spectator** (**GGZdMod** *ggzdmod, int spectator)

  Get a spectator's data.

- int **ggzdmod_set_handler** (**GGZdMod** *ggzdmod, **GGZdModEvent** e, **GGZdModHandler** func)

  Set a handler for the given event.

- int **ggzdmod_count_seats** (**GGZdMod** *ggzdmod, GGZSeatType seat_type)

  Count seats of the given type.

- int **ggzdmod_count_spectators** (**GGZdMod** *ggzdmod)

  Count current number of spectators.

- int **ggzdmod_dispatch** (**GGZdMod** *ggzdmod)

  Check for and handle input.

- int **ggzdmod_loop** (**GGZdMod** *ggzdmod)

  Loop while handling input.

- int **ggzdmod_set_state** (**GGZdMod** *ggzdmod, **GGZdModState** state)

  Change the table's state.

- int **ggzdmod_connect** (**GGZdMod** *ggzdmod)

  Connect to ggz.

- int **ggzdmod_disconnect** (**GGZdMod** *ggzdmod)

  Disconnect from ggz.

- int **ggzdmod_log** (**GGZdMod** *ggzdmod, const char *fmt,...) ggz__attribute((format(printf

  Log data.

- int void **ggzdmod_check** (**GGZdMod** *ggzdmod)

  Log all information about the ggzdmod.

- void **ggzdmod_report_game** (**GGZdMod** *ggzdmod, int *teams, **GGZGameResult** *results, int *scores)

  Report the results of a game to GGZ.

- void **ggzdmod_report_savegame** (**GGZdMod** *ggzdmod, const char *savegame)

  Report the savegame to GGZ.

- void **ggzdmod_request_num_seats** (**GGZdMod** *ggzdmod, int num_seats)

  Tell GGZ to change the number of seats at this table.

- void **ggzdmod_request_boot** (**GGZdMod** *ggzdmod, const char *name)

  Tell GGZ to boot the given player from this table.

- void **ggzdmod_request_bot** (**GGZdMod** *ggzdmod, int seat_num)

  Tell GGZ to change the given seat from OPEN to BOT.

- void **ggzdmod_request_open** (**GGZdMod** *ggzdmod, int seat_num)

  Tell GGZ to change the given seat from BOT/RESERVED to OPEN.

4.1.1 Detailed Description

Common functions for interfacing a game server and GGZ.

This file contains all libggzdmod functions used by game servers to interface with GGZ (and vice versa). Just include **ggzdmod.h** (p. 7) and make sure your program is linked with libggzdmod. Then use the functions below as appropriate.

GGZdmod currently provides an event-driven interface. Data from communication sockets is read in by the library, and a handler function (registered as a callback) is invoked to handle any events. The calling program should not read/write data from/to the GGZ socket unless it really knows what it is doing.

That this does not apply to the client sockets: ggzdmod provides one file desriptor for communicating (TCP) to each client. If data is ready to be read by one of these file descriptors ggzdmod may invoke the appropriate handler (see below), but will never actually read any data.

Here is a fairly complete example. In this game we register a handler for each of the possible callbacks. This particular game is played only when all seats are full; when any seats are empty it must wait (much like a card or board game).

```
// Game-defined handler functions for GGZ events; see below.
void handle_state_change(GGZdMod* ggz, GGZdModEvent event,
                         const void *data);
void handle_player_seat(GGZdMod* ggz, GGZdModEvent event,
                        const void *data);
void handle_player_data(GGZdMod* ggz, GGZdModEvent event,
```

```
                        const void *data);

// Other game-defined functions (not ggz-related).
void game_init(GGZdMod *ggz); // initialize a game
void game_launch(void);           // handle a game "launch"
void game_end(void);              // called before the table shuts down
void resume_playing(void);        // we have enough players to play
void stop_playing(void);          // not enough players to play

int main()
{
    GGZdMod *ggz = ggzdmod_new(GGZ_GAME);
    // First we register functions to handle some events.
    ggzdmod_set_handler(ggz, GGZDMOD_EVENT_STATE,
                        &handle_state_change);
    ggzdmod_set_handler(ggz, GGZDMOD_EVENT_JOIN,
                        &handle_player_seat);
    ggzdmod_set_handler(ggz, GGZDMOD_EVENT_LEAVE,
                        &handle_player_seat);
    ggzdmod_set_handler(ggz, GGZDMOD_EVENT_SEAT,
                        &handle_player_seat);
    ggzdmod_set_handler(ggz, GGZDMOD_EVENT_PLAYER_DATA,
                        &handle_player_data);

    // Do any other game initializations.  You'll probably want to
    // track "ggz" globally.
    game_init(mod);

    // Then we must connect to GGZ
    if (ggzdmod_connect(ggz) < 0)
        exit(-1);
    (void) ggzdmod_log(ggz, "Starting game.");

    // ggzdmod_loop does most of the work, dispatching handlers
    // above as necessary.
    (void) ggzdmod_loop(ggz);

    // At the end, we disconnect and destroy the ggzdmod object.
    (void) ggzdmod_log(ggz, "Ending game.");
    (void) ggzdmod_disconnect(ggz);
    ggzdmod_free(ggz);
}

void handle_state_change(GGZdMod* ggz, GGZdModEvent event,
                         const void *data)
{
    const GGZdModState *old_state = data;
    GGZdModState new_state = ggzdmod_get_state(ggz);
    if (*old_state == GGZDMOD_STATE_CREATED)
        // ggzdmod data isn't initialized until it connects with GGZ
        // during the game launch, so some initializations should wait
        // until here.
        game_launch();
    switch (new_state) {
      case GGZDMOD_STATE_WAITING:
        // At this point we've entered the "waiting" state where we
        // aren't actually playing.  This is generally triggered by
        // the game calling ggzdmod_set_state, which happens when
        // a player leaves (down below).  It may also be triggered
        // by GGZ automatically.
        stop_playing();
        break;
      case GGZDMOD_STATE_PLAYING:
        // At this point we've entered the "playing" state, so we
        // should resume play.  This is generally triggered by
        // the game calling ggzdmod_set_state, which happens when
        // all seats are full (down below).  It may also be
```

```
        // triggered by GGZ automatically.
        resume_playing();
        break;
      case GGZDMOD_STATE_DONE:
        // at this point ggzdmod_loop will stop looping, so we'd
        // better close up shop fast.  This will only happen
        // automatically if all players leave, but we can force it
        // using ggzdmod_set_state.
        game_end();
        break;
    }
}

void handle_player_seat(GGZdMod* ggz, GGZdModEvent event,
                        const void *data)
{
  const GGZSeat *old_seat = data;
  GGZSeat new_seat = ggzdmod_get_seat(ggz, old_seat->num);

  if (new_seat.type == GGZ_SEAT_PLAYER
      && old_seat->type != GGZ_SEAT_PLAYER) {
    // join event ... do player initializations ...

    if (ggzdmod_count_seats(ggz, GGZ_SEAT_OPEN) == 0) {
      // this particular game will only play when all seats are full.
      // calling this function triggers the STATE event, so we'll end
      // up executing resume_playing() above.
      ggzdmod_set_state(ggz, GGZDMOD_STATE_PLAYING);
    }
  } else if (new_seat.type != GGZ_SEAT_PLAYER
             && old_seat->type == GGZ_SEAT_PLAYER) {
    // leave event ... do de-initialization ...

    if (ggzdmod_count_seats(ggz, GGZ_SEAT_PLAYER) == 0)
        // the game will exit when all human players are gone
        ggzdmod_set_state(ggz, GGZDMOD_STATE_DONE);
    else
        // this particular game will only play when all seats are full.
        // calling this function triggers the STATE event, so we'll end
        // up executing stop_playing() above.
        ggzdmod_set_state(ggz, GGZDMOD_STATE_WAITING);
  }
}

void handle_player_data(GGZdMod* ggz, GGZdModEvent event,
                        const void *data)
{
    const int *player = data;
    int socket_fd = ggzdmod_get_seat(ggz, *player).fd;

    // ... read a packet from the socket ...
}
```

For more information, see the documentation at `http://www.ggzgamingzone.org/`.

4.1.2 Typedef Documentation

4.1.2.1 typedef struct GGZdMod GGZdMod

A GGZdmod object, used for tracking a ggzd<->table connection.

A game server should track a pointer to a GGZdMod object; it contains all the state information for communicating with GGZ. The GGZ server will track one such object for every game table that is running.

4.1.2.2 typedef void(*) GGZdModHandler(GGZdMod *mod, GGZdModEvent event, const void *data)

Event handler prototype.

A function of this type will be called to handle a ggzdmod event.

Parameters:

mod The ggzdmod state object.

event The event that has occured.

data Pointer to additional data for the event. The additional data will be of the following form:

- GGZDMOD_EVENT_STATE: The old state (GGZdModState*)
- GGZDMOD_EVENT_JOIN: The old seat (GGZSeat*)
- GGZDMOD_EVENT_LEAVE: The old seat (GGZSeat*)
- GGZDMOD_EVENT_SEAT: The old seat (GGZSeat*)
- GGZDMOD_EVENT_SPECTATOR_JOIN: The old spectator's data (GGZSeat*)
- GGZDMOD_EVENT_SPECTATOR_LEAVE: The old spectator's data (GGZSeat*)
- GGZDMOD_EVENT_LOG: The message string (char*)
- GGZDMOD_EVENT_PLAYER_DATA: The player number (int*)
- GGZDMOD_EVENT_SPECTATOR_DATA: The spectator number (int*)
- GGZDMOD_EVENT_ERROR: An error string (char*)

4.1.3 Enumeration Type Documentation

4.1.3.1 enum GGZdModEvent

Callback events.

Each of these is a possible GGZdmod event. For each event, the table may register a handler with GGZdmod to handle that event.

See also:

GGZdModHandler (p. 13)
ggzdmod_set_handler (p. 22)

Enumerator:

GGZDMOD_EVENT_STATE Module status changed This event occurs when the game's status changes. The old state (a GGZdModState*) is passed as the event's data.

See also:

GGZdModState (p. 14)

GGZDMOD_EVENT_JOIN Player joined This event occurs when a player joins the table. The old seat (a GGZSeat*) is passed as the event's data. The seat information will be updated before the event is invoked.

GGZDMOD_EVENT_LEAVE Player left This event occurs when a player leaves the table. The old seat (a GGZSeat*) is passed as the event's data. The seat information will be updated before the event is invoked.

GGZDMOD_EVENT_SEAT General seat change This event occurs when a seat change other than a player leave/join happens. The old seat (a GGZSeat*) is passed as the event's data. The seat information will be updated before the event is invoked. Operations include changing of reserved or abandoned seats to open, swapping bot players in and out, making a reservation on an open seat, or even an open seat changing to a player seat if a player changes seats. Note that no new connections are provided nor are connections removed in a SEAT event; thus, if a player is removed via this event you can be sure another SEAT or SPECTATOR_SEAT event will be provided shortly to re-add him to a new location.

GGZDMOD_EVENT_SPECTATOR_JOIN A spectator joins the game. The data of the old spectator (GGZSeat*) is passed as the data for the event. It can be assumed that the spectator seat was previously empty, so the name and socket given will be invalid (NULL/-1).

GGZDMOD_EVENT_SPECTATOR_LEAVE A spectator left the game The old spectator data can be obtained via the (GGZSeat*) which is passed as the event data.

GGZDMOD_EVENT_SPECTATOR_SEAT A spectator seat changed. The old spectator data can be obtained via the (GGZSeat*) which is passed as the event data. The same caveats apply as to GGZDMOD_EVENT_SEAT.

See also:

GGZDMOD_EVENT_SEAT (p. 14)

GGZDMOD_EVENT_PLAYER_DATA Data available from player This event occurs when there is data ready to be read from one of the player sockets. The player number (an int*) is passed as the event's data.

GGZDMOD_EVENT_SPECTATOR_DATA Data available from spectator For games which support spectators, this indicates that one of them sent some data to the game server.

GGZDMOD_EVENT_ERROR An error has occurred This event occurs when a GGZdMod error has occurred. An error message (a char*) will be passed as the event's data. GGZdMod may attempt to recover from the error, but it is not guaranteed that the GGZ connection will continue to work after an error has happened.

4.1.3.2 enum GGZdModState

Table states.

Each table has a current "state" that is tracked by ggzdmod. First the table is executed and begins running. Then it receives a launch event from GGZD and begins waiting for players. At some point a game will be started and played at the table, after which it may return to waiting. Eventually the table will probably halt and then the program will exit.

More specifically, the game is in the CREATED state when it is first executed. It moves to the WAITING state after GGZ first communicates with it. After this, the game server may use ggzdmod_set_state to change between WAITING, PLAYING, and DONE states. A WAITING game is considered waiting for players (or whatever), while a PLAYING game is actively being played (this information may be, but currently is not, propogated back to GGZ for display purposes). Once the state is changed to DONE, the table is considered dead and will exit shortly thereafter (ggzdmod_loop will stop looping, etc.) (see the kill_on_exit game option).

Each time the game state changes, a GGZDMOD_EVENT_STATE event will be propogated to the game server.

Enumerator:

GGZDMOD_STATE_CREATED Pre-launch; waiting for ggzdmod

GGZDMOD_STATE_WAITING Ready and waiting to play.

GGZDMOD_STATE_PLAYING Currently playing a game.

GGZDMOD_STATE_DONE Table halted, prepping to exit.

4.1.3.3 enum GGZdModType

The "type" of ggzdmod.

The "flavor" of GGZdmod object this is. Affects what operations are allowed.

Enumerator:

GGZDMOD_GGZ Used by the ggz server ("ggzd").

GGZDMOD_GAME Used by the game server ("table").

4.1.3.4 enum GGZGameResult

Enumerator:

GGZ_GAME_FORFEIT A forfeit is (for instance) an abandoned game. The player will not only be credited with the forfeit but their rating/ranking may drop dramatically.

GGZ_GAME_NONE If the player didn't take part in the game, use this label. For instance if one player abandons the game they might get a forfeit while nobody else is affected.

4.1.4 Function Documentation

4.1.4.1 int void ggzdmod_check (GGZdMod * *ggzdmod*)

Log all information about the ggzdmod.

This is a debugging function that will log all available information about the GGZdMod object. It uses ggzdmod_log for logging.

Parameters:

ggzdmod The GGZdMod object.

Returns:

void; errors in ggzdmod_log are ignored.

4.1.4.2 int ggzdmod_connect (GGZdMod * *ggzdmod*)

Connect to ggz.

Call this function to make an initial GGZ <-> game connection.

- When called by the game server, this function makes the physical connection to ggz.

- When called by ggzd, it will launch a table and connect to it. Note - if the game fails to exec, this function may not catch it.

  Parameters:

  ggzdmod The ggzdmod object.

  Returns:

  0 on success, -1 on failure.

4.1.4.3 int ggzdmod_count_seats (GGZdMod * *ggzdmod*, GGZSeatType *seat_type*)

Count seats of the given type.

This is a convenience function that counts how many seats there are that have the given type. For instance, giving seat_type==GGZ_SEAT_OPEN will count the number of open seats.

Parameters:

ggzdmod The ggzdmod object.

seat_type The type of seat to be counted.

Returns:

The number of seats that match seat_type.

Note:

This could go into a wrapper library instead.

4.1.4.4 int ggzdmod_count_spectators (GGZdMod * *ggzdmod*)

Count current number of spectators.

This function returns the number of spectators watching the game. Note that the spectator numbers may not match up: if there are two spectators they could be numbered 0 and 4. If you're trying to iterate through the existing spectators, you probably want **ggzdmod_get_max_num_spectators()** (p. 18) instead.

Parameters:

ggzdmod The ggzdmod object

Returns:

The number of spectators watching the game (0 on error)

4.1.4.5 int ggzdmod_disconnect (GGZdMod * *ggzdmod*)

Disconnect from ggz.

- When called by the game server, this function stops the connection to GGZ. It should only be called when the table is ready to exit.

- When called by the GGZ server, this function will kill and clean up after the table.

  Parameters:

  ggzdmod The ggzdmod object.

  Returns:

  0 on success, -1 on failure.

4.1.4.6 int ggzdmod_dispatch (GGZdMod * *ggzdmod*)

Check for and handle input.

This function handles input from the communications sockets:

- It will check for input, but will not block.
- It will monitor input from the GGZdmod socket.
- It will monitor input from player sockets only if a handler is registered for the PLAYER_-DATA event.
- It will call an event handler as necessary.

  Parameters:

  ggzdmod The ggzdmod object.

  Returns:

  -1 on error, the number of events handled (0 or more) on success.

4.1.4.7 void ggzdmod_free (GGZdMod * *ggzdmod*)

Destroy a finished ggzdmod object.

After the connection is through, the object may be freed.

Parameters:

ggzdmod The GGZdMod object.

4.1.4.8 char* ggzdmod_get_bot_class (GGZdMod * *ggzdmod*, const char * *name*)

Return class for named bot.

Parameters:

ggzdmod The GGZdMod object.

name Name of the bot.

Returns:

The bot's class, or NULL for anonymous bots.

4.1.4.9 int ggzdmod_get_fd (GGZdMod * *ggzdmod*)

Get the file descriptor for the GGZdMod socket.

Parameters:

ggzdmod The GGZdMod object.

Returns:

GGZdMod's main ggzd <-> table socket FD.

Note:

Don't use this; use ggzdmod_loop and friends instead.

4.1.4.10 void* ggzdmod_get_gamedata (GGZdMod * *ggzdmod*)

Return gamedata pointer.

Each GGZdMod object can be given a "gamedata" pointer that is returned by this function. This is useful for when a single process serves multiple GGZdmod's.

Parameters:

ggzdmod The GGZdMod object.

Returns:

A pointer to the gamedata block (or NULL if none).

See also:

ggzdmod_set_gamedata (p. 22)

4.1.4.11 int ggzdmod_get_max_num_spectators (GGZdMod * *ggzdmod*)

Get the maximum number of spectators. This function returns the maximum number of spectator seats available. A game can use this to iterate over the spectator seats to look for spectators occupying them. Since spectators may come and go at any point and there is no theoretical limit on the number of spectators, you should consider this value to be dynamic and call this function again each time you're looking for spectators.

Returns:

The number of available spectator seats, or -1 on error.

Note:

If no connection is present, -1 will be returned.

4.1.4.12 int ggzdmod_get_num_seats (GGZdMod * *ggzdmod*)

Get the total number of seats at the table.

Returns:

The number of seats, or -1 on error.

Note:

If no connection is present, -1 will be returned.
While in GGZDMOD_STATE_CREATED, we don't know the number of seats.

4.1.4.13 GGZSeat ggzdmod_get_seat (GGZdMod * *ggzdmod*, int *seat*)

Get all data for the specified seat.

Parameters:

ggzdmod The GGZdMod object.

seat The seat number (0..(number of seats - 1)).

Returns:

A valid **GGZSeat** (p. 5) structure, if seat is a valid seat.

4.1.4.14 GGZSeat ggzdmod_get_spectator (GGZdMod * *ggzdmod*, int *spectator*)

Get a spectator's data.

Parameters:

ggzdmod The GGZdMod object.

spectator The number, between 0 and (number of spectators - 1).

Returns:

A valid **GGZSeat** (p. 5) structure, if arguments are valid.

4.1.4.15 GGZdModState ggzdmod_get_state (GGZdMod * *ggzdmod*)

Get the current state of the table.

Parameters:

ggzdmod The GGZdMod object.

Returns:

The state of the table.

4.1.4.16 GGZdModType ggzdmod_get_type (GGZdMod * *ggzdmod*)

Get the type of the ggzdmod object.

Parameters:

ggzdmod The GGZdMod object.

Returns:

The type of the GGZdMod object (GGZ or GAME).

4.1.4.17 int ggzdmod_is_ggz_mode (void)

Is the program running in GGZ mode?

Call this function to see if the program was actually launched by GGZ. This can be used to give an error message if the executable is run outside of the GGZ environment, or for games that will run both inside and outside of GGZ.

Returns:

A boolean value indicating whether the program is running in GGZ.

Note:

Should only be called by game servers, not by GGZ itself.

4.1.4.18 int ggzdmod_log (GGZdMod * *ggzdmod*, const char * *fmt*, ...)

Log data.

This function sends the specified string (printf-style) to the GGZ server to be logged.

Parameters:

ggzdmod The GGZdmod object.

fmt A printf-style format string.

Returns:

0 on success, -1 on failure.

4.1.4.19 int ggzdmod_loop (GGZdMod * *ggzdmod*)

Loop while handling input.

This function repeatedly handles input from all sockets. It will only stop once the game state has been changed to DONE (or if there has been an error).

Parameters:

ggzdmod The ggzdmod object.

Returns:

0 on success, -1 on error.

See also:

ggzdmod_dispatch (p. 17)
ggzdmod_set_state (p. 23)

4.1.4.20 GGZdMod* ggzdmod_new (GGZdModType *type*)

Create a new ggzdmod object.

Before connecting through ggzdmod, a new ggzdmod object is needed.

Parameters:

type The type of ggzdmod. Should be GGZDMOD_GAME for game servers.

See also:

GGZdModType (p. 15)

4.1.4.21 void ggzdmod_report_game (GGZdMod * *ggzdmod*, int * *teams*, GGZGameResult * *results*, int * *scores*)

Report the results of a game to GGZ.

After a game has completed, the game server should call this function to report the results to GGZ. GGZ can then use the information to track player statistics - including an ELO-style rating, win-loss records, etc.

Parameters:

ggzdmod The ggzdmod object.

teams An array listing a team number for each player, or NULL.

results An array listing the result of the game for each player.

scores The scores for all players (may be NULL)

4.1.4.22 void ggzdmod_report_savegame (GGZdMod * *ggzdmod*, const char * *savegame*)

Report the savegame to GGZ.

If a game saves the game data to disk, the directory name, file name or any other associated token can be reported to GGZ. In the case of a continuous game log, the reporting should happen at the beginning as to allow the continuation of the saved game.

Parameters:

ggzdmod The ggzdmod object.

savegame Name of the savegame file within the game's directory.

4.1.4.23 void ggzdmod_request_num_seats (GGZdMod * *ggzdmod*, int *num_seats*)

Tell GGZ to change the number of seats at this table.

Note:

This functionality is incomplete, and should not yet be used.

4.1.4.24 void ggzdmod_set_gamedata (GGZdMod * *ggzdmod*, void * *data*)

Set gamedata pointer.

Parameters:

ggzdmod The GGZdMod object.

data The gamedata block (or NULL for none).

See also:

ggzdmod_get_gamedata (p. 18)

4.1.4.25 int ggzdmod_set_handler (GGZdMod * *ggzdmod*, GGZdModEvent *e*, GGZdModHandler *func*)

Set a handler for the given event.

As described above, GGZdmod uses an event-driven structure. Each time an event is called, the event handler (there can be only one) for that event will be called. This function registers such an event handler.

Parameters:

ggzdmod The GGZdmod object.

e The GGZdmod event.

func The handler function being registered.

Returns:

0 on success, negative on failure (bad parameters)

See also:

ggzdmod_get_gamedata (p. 18)

4.1.4.26 void ggzdmod_set_playerdata (GGZdMod * *ggzdmod*, bool *is_spectator*, int *seat_num*, void * *playerdata*)

Set a playerdata pointer.

Each GGZ seat (regular or spectator) can be given a "playerdata" pointer that is available through ggzdmod_get_seat. This is useful for preserving state data when a particular player changes seat, as the playerdata will be preserved across the seat change.

Parameters:

ggzdmod The GGZdMod object.

is_spectator true iff it is a spectator seat

seat_num The number of the seat

playerdata An arbitrary pointer to be set as the playerdata

4.1.4.27 int ggzdmod_set_state (GGZdMod * *ggzdmod*, GGZdModState *state*)

Change the table's state.

This function should be called to change the state of a table. A game can use this function to change state between WAITING and PLAYING, or to set it to DONE.

Parameters:

ggzdmod The ggzdmod object.

state The new state.

Returns:

0 on success, -1 on failure/error.

Index