Server Interface

# Code Request Summary

The server deciphers each request through a series of codes:

// Global user settings

DEFAULT\_USER\_LEVEL = 0;

// Server settings

SERVER\_PORT = 80;

// Actions

REGISTER\_ACTION = 100;

LOGIN\_ACTION = 101;

PROFILE\_ACTION = 102;

PROFILE\_UPDATE = 103;

UNKNOWN\_ACTION = 199;

FRIEND\_GET = 500;

FRIEND\_SEARCH = 503;

FRIEND\_ADD = 506;

FRIEND\_CHECK = 509;

FRIEND\_DELETE = 512;

MESSAGE\_SEND = 600;

MESSAGE\_RECEIVE = 601;

MESSAGE\_OFFLINE\_GET = 605;

MESSAGE\_COMMAND\_SEND = 606;

MESSAGE\_LOCATION\_SEND = 610;

MESSAGE\_LOCATION\_RECEIVE = 613;

GAME\_CREATE = 700;

GAME\_ADD = 703;

GAME\_EXIT = 706;

GAME\_GET = 709;

GAME\_DELETE = 712;

GAME\_USER\_REMOVE = 715;

GAME\_GET\_CURRENT = 718;

GAME\_NOTIFICATION\_SEND = 721;

GAME\_NOTIFICATION\_RECEIVE = 722;

GAME\_GET\_USER = 723;

// Default status

MESSAGE\_READ = 1;

MESSAGE\_UNREAD = 0;

GAME\_OWNER = 1;

GAME\_PLAYER = 0;

// Response code

PROFILE\_UPDATE\_SUCCESS = 104;

PROFILE\_UPDATE\_FAIL = 105;

LOGIN\_SUCCESS\_CODE = 200;

LOGIN\_USER\_NON\_EXIST\_CODE = 201;

LOGIN\_EXIST\_CODE = 202;

PROFILE\_GET = 203;

RESISTER\_SUCCESS = 300;

REGISTER\_FAIL = 301;

FRIEND\_GET\_FAIL = 501;

FRIEND\_GET\_SUCCESS = 502;

FRIEND\_SEARCH\_FAIL = 504;

FRIEND\_SEARCH\_SUCCESS = 505;

FRIEND\_ADD\_FAIL = 507;

FRIEND\_ADD\_SUCCESS = 508;

FRIEND\_CHECK\_FAIL = 510;

FRIEND\_CHECK\_SUCCESS = 511;

FRIEND\_DELETE\_SUCCESS = 513;

FRIEND\_DELETE\_FAIL = 514;

MESSAGE\_SEND\_SUCCESS\_ONLINE = 602;

MESSAGE\_SEND\_SUCCESS\_OFFLINE = 603;

MESSAGE\_SEND\_FAIL = 604;

MESSAGE\_COMMAND\_SUCCESS = 607;

MESSAGE\_COMMAND\_FAIL = 608;

MESSAGE\_COMMAND\_RECEIVE = 609;

MESSAGE\_LOCATION\_SUCCESS = 611;

MESSAGE\_LOCATION\_FAIL = 612;

GAME\_CREATE\_SUCCESS = 701;

GAME\_CREATE\_FAIL = 702;

GAME\_ADD\_SUCCESS = 704;

GAME\_ADD\_FAIL = 705;

GAME\_EXIT\_SUCCESS = 707;

GAME\_EXIT\_FAIL = 708;

GAME\_GET\_SUCCESS = 710;

GAME\_GET\_FAIL = 711;

GAME\_DELETE\_SUCCESS = 713;

GAME\_DELETE\_FAIL = 714;

GAME\_USER\_REMOVE\_SUCCESS = 716;

GAME\_USER\_REMOVE\_FAIL = 717;

GAME\_GET\_CURRENT\_SUCCESS = 719;

GAME\_GET\_CURRENT\_FAIL = 720;

GAME\_GET\_USER\_SUCCESS = 724;

GAME\_GET\_USER\_FAIL = 725;

# Master Functions

## Send to all other users

Send a message to all users. This function needs user level > 0.

Request

{

“action”: 1001

“message”: string

}

Return

Nothing

# Notes

1. Both sending && receiving are using JSON format, other format will get no response
2. Sample request may have incorrect code of response, all correct codes are listed above or below each subtitle.

# User Online Checking Service

Check whether the user is online.

FRIEND\_CHECK = 509;

FRIEND\_CHECK\_FAIL = 510;

FRIEND\_CHECK\_SUCCESS = 511;

Request

{

Username: string

action: int

}

Return

{

Status: string

Code: int

}

Example Request

Send

{

“username”: 1

“action”: 509

}

Return

{

"status":"success",

"code":510

}

# Server Login Service

The interface between the server and client which allows clients send login request to server.

When user logs in successfully, the user will be assigned a token, which is a unique ID of current session. Whenever the user sends an action to the server, the token must be attached as the authentication.

LOGIN\_ACTION = 101;

// Send the log in request to server

LOGIN\_SUCCESS\_CODE = 200;

// Log in success

LOGIN\_USER\_NON\_EXIST\_CODE = 201;

// Log in failed, because the user does not exist

LOGIN\_EXIST\_CODE = 202;

// Log in failed, because the user has logged in.

Message

{

username: string,

password: string,

action: int

}

Return

{

status: string,

code: int,

}

Example Request

Send

{

“username”: “test”,

“password”: “donttellothers”,

“action”: 101

}

Return

{

"status": "success",

"code": 200

}

# Client Register Service

The interface between the server and client which allows clients to create a new account in database.

REGISTER\_ACTION = 100;

RESISTER\_SUCCESS = 300;

REGISTER\_FAIL = 301;

Message

{

username: string,

password: string,

email: string,

name: string,

action: int

}

Return

{

status: string,

code: int

}

Example Request

Send

{

“username”: “test”,

“password”: “donttellothers”,

“email”: “[auser@catchmeifyoucan.com](mailto:auser@catchmeifyoucan.com)”,

“name”: “tester”,

“action”: 100

}

Return

{

"status": "success",

"code": 300

}

# Get Profile Service

Get user’s profile. User must log in first.

Request

{

action: int

}

Return

{

action: string

name: string

email: string

username: string

id: int

lv: int

location: string

status: string

}

Example Request

Send

{

“action”: 102

}

Return

{"action":203,"name":"","username":"admin","email":"","id":1,"lv":0,"location":"","status":""}

# Update Profile Service

Update user’s profile. User must log in first.

Request

{

action: int

name: string

email: string

location: string

status: string

}

Return

{

status: string

code: int

}

Example Request

Send

{"action":103,"name":"222","email":"haha@uu.xx”,"location":"Mel","status":"GLHF"}

Return

{"status":"updates success","code":104}

# Add Friend Service

Add a friend using the friend’s username.

FRIEND\_ADD = 506;

FRIEND\_ADD\_FAIL = 507;

FRIEND\_ADD\_SUCCESS = 508;

Request

{

Username: string

action: int

}

Return

{

Status: string

Code: int

}

Example Request

Send

{

“username”: 1

“action”: 506

}

Return

{

"status":"success",

"code":508

}

# Search User Service

Search a user’s profile and ready to add the user as a friend.

FRIEND\_SEARCH = 503;

FRIEND\_SEARCH\_FAIL = 504;

FRIEND\_SEARCH\_SUCCESS = 505;

Request

{

Username: string

action: int

}

Return

{

Status: string

Code: int

Result: array that include the user information

}

Example Request

Send

{

“username”: “1”,

“action”: 503

}

Return

{

"status":"success",

"code":502,

"result":[{"username":"1", “email”:”xxx”,”name”:”asd”}]

}

# Fetch Friend List Request Service

The interface between the server and client which allows clients to ask the server to send a JSON list of all the user’s friends.

FRIEND\_GET = 500;

FRIEND\_GET\_FAIL = 501;

FRIEND\_GET\_SUCCESS = 502;

Request

{

action: int

}

Return

{

Status: string

Code: int

Result: array

}

Example Request

Send

{

“action”: 500

}

Return

{

"status":"success",

"code":502,

"result":[

{"username":"1"},

{"username":"3"},

{"username":"admin3"},

{"username":"pls"},

{"username":"serious"}

]

}

# Create Game Service

Create a new game.

## Sending Format

{"action": int, “name”: string}

## Return Format

{"status": string, "code": int, “game\_id”: int}

## Sample Send

## {"action":700,"name":"hello??"}

## Sample Return

{"status":"success","code":701,"game\_id":18}

# Join Game Service

Join a game.

## Sending Format

{"action": int, “id”: int}

## Return Format

{"status": string, "code": int}

## Sample Send

## {"action":703, "id":19}

## Sample Return

{"status":"success","code":704}

# Exit Game Service

Exit current game. If the user is the owner of game(s), those game(s) will also be deleted, and will also return “game user removing” status.

GAME\_EXIT = 706;

GAME\_EXIT\_SUCCESS = 707;

GAME\_EXIT\_FAIL = 708;

GAME\_DELETE = 712;

GAME\_DELETE\_SUCCESS = 713;

GAME\_DELETE\_FAIL = 714;

GAME\_USER\_REMOVE = 715;

GAME\_USER\_REMOVE\_SUCCESS = 716;

GAME\_USER\_REMOVE\_FAIL = 717;

## Sending Format

{"action": int}

## Return Format

{"status": string, "code": int}

## Sample Send

{"action":706}

## Sample Return

{"status":"success","code":716}

{"status":"success","code":716}

{"status":"success","code":713}

# Remove Game User Service

TO BE CONTINUE.

GAME\_USER\_REMOVE = 715;

GAME\_USER\_REMOVE\_SUCCESS = 716;

GAME\_USER\_REMOVE\_FAIL = 717;

# Get Game Notification Service

Get notification of the game.

## Receive:

{

action: int

message: string

}

## Sample:

{"action":722,"message":"1 joined the game."}

# Send Game Notification Service

This will be sent automatically to all online users. This has no returns.

Message

{

action: int

message: string

}

Example Request

Send

{"action":721, "message": "12312321"}

# Fetch Gaming Users Service

The all users that currently in the game.

GAME\_GET\_USER = 723;

GAME\_GET\_USER\_SUCCESS = 724;

GAME\_GET\_USER\_FAIL = 725;

Request

{

action: int

}

Return

{

Status: string

Code: int

Result: array

}

Example Request

Send

{

“action”: 723

}

Return

{

"status":"success",

"code":724,

"result":[

{"account\_id":1,” game\_id”:1, “is\_owner”: 1},

{"account\_id":2,” game\_id”:1, “is\_owner”: 0}

]

}

# Fetch Games Service

Get all games.

GAME\_GET = 709;

GAME\_GET\_SUCCESS = 710;

GAME\_GET\_FAIL = 711;

Request

{

action: int

}

Return

{

Status: string

Code: int

Result: array

}

Example Request

Send

{

“action”: 709

}

Return

{

"status":"success",

"code":710,

"result":[

{“game\_id”:1, “name”: “name1”},

{“game\_id”:2, “name”: “name2”}

]

}

# Fetch Current Games Service

This helps those who reconnect to the server and want to rejoin the game.

GAME\_GET\_CURRENT = 718;

GAME\_GET\_CURRENT\_SUCCESS = 719;

GAME\_GET\_CURRENT\_FAIL = 720;

Request

{

action: int

}

Return

{

Status: string

Code: int

Result: array

}

Example Request

Send

{

“action”: 718

}

Return

{

"status":"success",

"code":719,

"result":[

{“game\_id”:1, “name”: “name1”},

]

}

# Send Location to Game Creator

Send players location to game creator in the game.

Request

{

action: int

location: {JSON format location}

}

Return

{

Status: string

Code: int

}

Example Request

Send

{

“action”: 610

“location":{"x":1,"y":2}

}

Return

{"status":"message delivered","code":611}

# Creator Receive Location

The location information will delivery directly to the creator, or fails. Therefore, the request from creator is not needed.

Request

None

Return

{

action: int

location: {JSON location}

}

Example Request

Return

{"action":613,"location":{"x":1,"y":2}}

# User Message Sending Service (Command)

User can send commands to other users.

MESSAGE\_COMMAND\_SEND = 606;

MESSAGE\_COMMAND\_SUCCESS = 607;

MESSAGE\_COMMAND\_FAIL = 608;

MESSAGE\_COMMAND\_RECEIVE = 609;

Message

{

action: int

username: string

message: string

}

Return

{

status: string,

code: int

}

Example Request

Send

{

“action”: 606

“username”: “zirenxiao”

“message”: “have you seen my beer Tibbers?”

}

Return

{

"status": "success",

"code": 607

}

# User Message Receiving Service (Command)

The command will automatically deliver to online users.

## Receive:

{

action: int

message: string

from: string

}

## Sample:

{

“action”: 601,

“message”: “hahahah”,

“from”: “admin”

}

# Server Message Sending Service (Chat)

The interface between the server and client which allows clients send message to others. The user must log in first, or will get nothing response.

MESSAGE\_SEND = 600;

// send a message request to server

MESSAGE\_RECEIVE = 601;

// server send a message to you where the message is from another user

MESSAGE\_SEND\_SUCCESS\_ONLINE = 602;

// message send success to an online user

MESSAGE\_SEND\_SUCCESS\_OFFLINE = 603;

// message send success to an offline user

MESSAGE\_SEND\_FAIL = 604;

// message send failed to a user

MESSAGE\_OFFLINE\_GET = 605;

// send “get offline message” request to server

Message

{

action: int

username: string

message: string

}

Return

{

status: string,

code: int

}

Example Request

Send

{

“action”: 600

“username”: “zirenxiao”

“message”: “have you seen my beer Tibbers?”

}

Return

{

"status": "success",

"code": 602

}

# Server (Both Offline/Online) Message Receiving Service

The offline message is stored in database, and after the client logged in, it should check offline message first.

The offline message will send one-by-one.

If the user is online, the message will deliver directly to user without any request. The online message has the same format as the offline message.

Request

{

action: int

}

Return

{

action: int

message: string

from: string

}

Example Request

Send

{

“action”: 605

}

Return

{

“action”: 601,

“message”: “hahahah”,

“from”: “admin”

}