Research On GameSpy Protocol

Arves100, xiaojiuwo

First Edition

May 10, 2020

Contents

Part I

Introduction

Chapter 1

History of GameSpy

Chapter 2

Related Works

Part II

General Information

In this chapter we describe the structure of GameSpy SDK and GameSpy servers.

Chapter 3

SDK Module

GameSpy SDK contains of 16 modules.

- Brigades
- Chat
- Presence & Messaging
- CDKey
- Stats & Tracking
- Persistent Storage
- Transport
- NAT Negotation
- Peer to Peer communication
- Patching & Tracking
- Server Browser
- Query & Reporting
- SAKE Persistent Storage
- ATLAS Competition
- Voice Chat
- Web Authentication

Chapter 4

GameSpy Back-end Servers

GameSpy back-end servers are list as follows.

- GameSpy Presence Connection Manager (GPCM)
- GameSpy Presence Search Player(GPSP)
- GameSpy Query and Report (QR)
- GameSpy Server Browser (SB)
- GameSpy Stats & Tracking (GStats)
- GameSpy Chat
- GameSpy NAT Negotation (NatNeg)
- GameSpy CDKey
- GameSpy Web Services
- GameSpy SAKE Storage (SAKE)

Chapter 5

Access Sequence of The Client

If a user want to use GameSpy service, the access sequence is listed in Figure 5.1 and we describe the detail below.

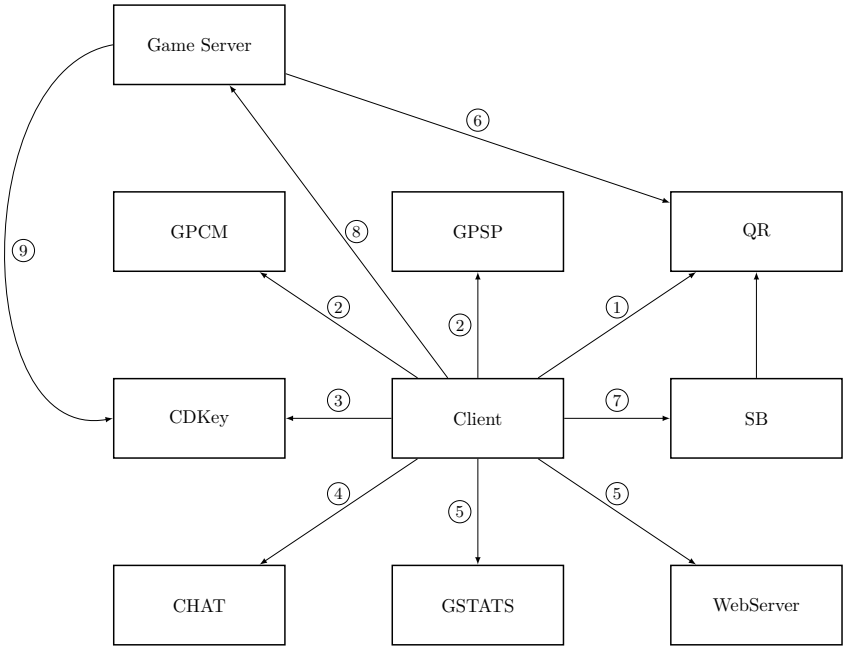


Figure 5.1: The access sequence of client

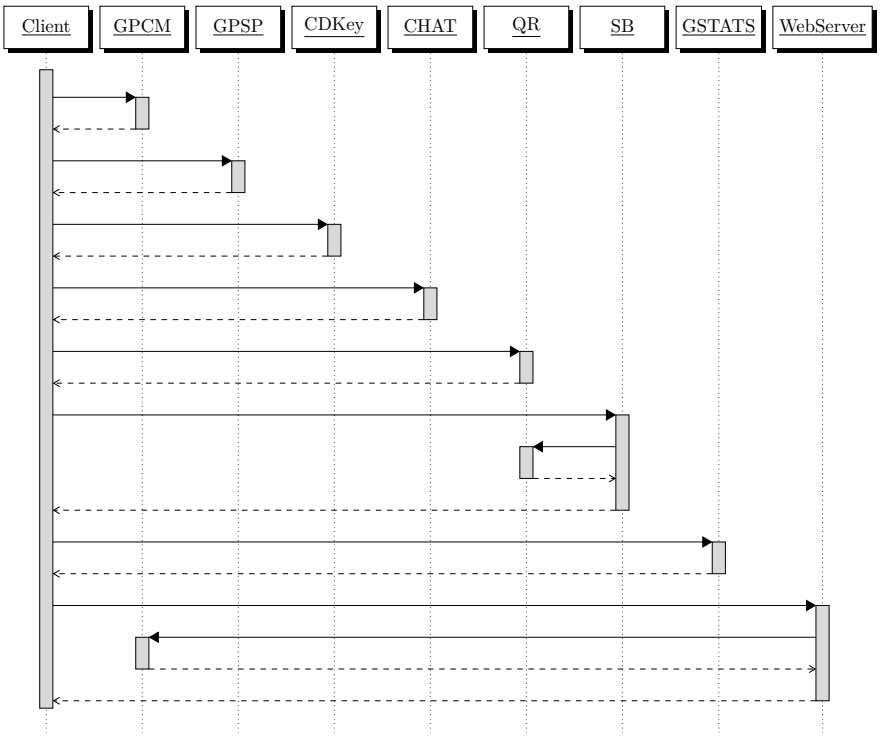


Figure 5.2: Login diagram

Explanation of access sequence

1. Client checks in QR server, which tells client GameSpy back-end server status.
2. Client accesses GPCM or GPSP to check their account and login.
3. Client accesses to CDKey to verify his cd-key in login phase.
4. Client logins to Chat server.
5. Client retrieves player data(level, exp, etc.) from GStats(old game use this server to store player data, new game use Web Server to store player data).
6. When a game server is launched it will send heartbeat to QR server to tell QR its information.
7. Client accesses to SB to search online game server.
8. Client logins to game server with his information and cd-key.
9. Game server will check his cd-key by accessing to CDKey server, after every information is verified, client should be able to play their game.

Chapter 6

Basic Description of Protocol

In this part, we describe some of the basic patterns that are used in all GameSpy servers.

6.1 String Pattern

We first introduce the pattern of the string, which is used to make up a request and response. The following servers do use the pattern: Presence Connection Manager, Presence Search Player, GameSpy Status and Tracking, CD-Key, Query Report(version 1) This kind of string represents a value in a request and response sent by the client or the server as Table 6.1.

String	Description
key**value**\	The key is **key**, the value of the key is **value**

Table 6.1: String pattern

There are two kind of patterns the first one is value string, the second one is command string. **Value String** This kind of string represents a key value pair in the request or response string, it has a key and a correspond value as shown in Table 6.2.

String	Description
\pid\13\	The key is **pid**, the value of the **pid** is 13
\userid\0\	The key is **userid**, the value of the **userid** is 0

Table 6.2: Value string

Command String
This kind of string represents a command in a request sends by the client or the server as Table 6.3. The command will end with \\ or \ depends on whether run at the server-side or client-side.

String	Description
\command\\	This is a command

Table 6.3: Command string

Part III

GameSpy Presence & Messaging

Presence & Messaging system allows a game to add account authentication or registration, which includes a profile where personal information could be stored (such as email, first name), a friend list (called buddies), private messages.

GameSpy Presence contains two servers, GameSpy Presence Connection Manager (GPCM) and GameSpy Presence Search Player (GPSP). GPCM is a server that manages the profiles (such as login, storing the profile information).

Chapter 7

Common Information

In this section we describe the common information, methods, techniques that GPCM and GPSP have.

7.1 Server IP and Ports

Table 7.1 are the IP and Ports of GPCM and GPSP that client or game connect to.

Name	IP	Port
GPCM	gpcm.gamespy.com	29900 (tcp)
GPSP	gpsp.gamespy.com	29901 (tcp)

Table 7.1: IP and Ports for GameSpy Presence Servers

Chapter 8

GameSpy Presence Connection Manager

8.1 Request Command of GameSpy Presence Connection Manager

Table 8.1 lists the request (known by us) that clients send to GameSpy Presence Connection Manager server (GPCM).

Commands	Description
inviteto	Invite friends
login	Login to GPCM
getprofile	Get the profile of a player (including your own)
addbuddy	Add a player to my friend list
delbuddy	Delete a player from my friend list
updateui	Update login information (email, password)
updatepro	Update my profile such as first name, last name, gender etc.
logout	Logout manually by user
status	Update the status of a user (Such as what game is the player playing)
ka	Keep client or session alive
bm	Message command
blk	Block list
bdy	Friend list
lt	Login ticket

Table 8.1: Request For GameSpy Presence Connection Manager

Error response string for (GPCM, GPSP):

$$\backslash error\backslash\backslash err\backslash < errorcode > \backslash fatal\backslash\backslash errmsg\backslash < errormessage > \backslash id\backslash 1\backslash final\backslash \tag{8.1}$$

8.2 GPI Connect Module

8.2.1 Login

We show the login communication diagram in Fig 8.1

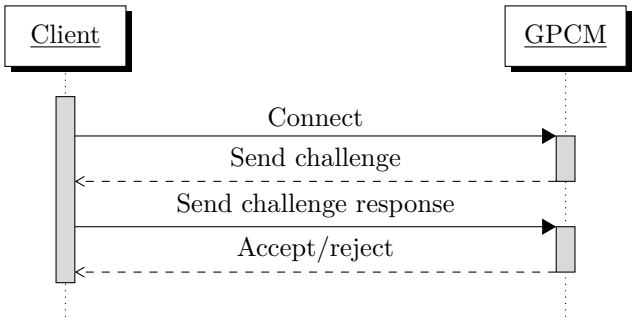


Figure 8.1: Login diagram

Server initial Challenge:

When a client is connected to GPCM server, GPCM Server will send a challenge to client. The challenge string shows in 8.2.1 and 8.2.2. However we do not know the correct functionality of 8.2.2.

Code 8.2.1

```
\lc\1\challenge\<challenge string>\final\
```

Code 8.2.2

```
\lc\1\challenge\<challenge string>\nur\\userid\<user id>
\profileid\<profile id>\final\
```

- challenge: The challenge string sent by GPCM.

Keys	Description	Type
challenge	The challenge string sended by GameSpy Presence server	String
nur	? Create new user delimiter	
userid	The userID of the profile	Uint
profileid	The profileID	Uint

Table 8.2: The first type login response

Client Login Request:

There are three ways of login:

- AuthToken: Logging using an alphanumeric string that represents an user.

- UniqueNick: Logging using a nickname that is unique from all the players.
- User: Logging with nickname, email and password.

We show the common part of login request in 8.2.3

Code 8.2.3

```
\login\\challenge\<challenge string>\⋆\userid\<user id>
\profileid\<profile id>\partnerid\<partner id>
\response\<challenge response string>\firewall\<firewall flag>
\port\<port>\productid\<product id>\gamename\<game name>
\sdkrevision\<sdk revision number>\quiet\<quiet mode flag>
\id\<operation id>\final\
```

Where the value of ⋆ in 8.2.3 depending on which login method user is using.

Code 8.2.4

```
\authtoken\<authentication token>\
\uniquenick\<uniquenick name>\
\user\<nick name+@+email>\
```

Keys	Description	Type
login	The login command which use to identify the login request of client	
challenge	The user challenge used to verify the authenticity of the client	See A
authtoken	The token used to login (represent of an user)	String
uniquenick	The unique nickname used to login	String
user	The users account (format is NICKNAME@EMAIL)	String
userid	User id	Uint
profileid	Profile id	Uint
partnerid	This ID is used to identify a backend service logged with gamespy.(Nintendo WIFI Connection will identify his partner as 11, which means that for gamespy, you are logging from a third party connection)	Uint
response	The client challenge used to verify the authenticity of the client	String
firewall	If this option is set to 1, then you are connecting under a firewall/limited connection	Uint
port	The peer port (used for p2p stuff)	Uint
productid	An ID that identify the game you're using	Uint
gamename	A string that rapresents the game that you're using, used also for several activities like peerchat server identification	string
namespaceid	Distinguish same nickname player	Uint
sdkrevision	The version of the SDK you're using	Uint
quiet	quite flag mode used in status buddy info 8.3.2	Uint
lt	The login ticket used for login into SAKE	String 25
id	The operation number	Uint

Table 8.3: Login parameter string

Server response:

When received client's login request, server check the challenge and proof. if client pass the check, server will first send response8.2.5 and then it will send friend list friend status, message, add friend request.

Code 8.2.5

\lc\2\sesskey\<session key>\userid\<user id>\uniquenick\<unique nick>\lt\<login ticket>\<challenge proof>\final\

Keys	Description	Type
sesskey	The session key, which is a integer rapresentating the client connection	Uint
userid	The userID of the profile	Uint
profileid	The profileID	Uint
uniquenick	The logged in unique nick	String
lt	The login ticket, unknown usage	String
proof	The proof is something similar to the response but it vary	String

Table 8.4: The second type login response

Proof in 8.4 generation: $md5(password)||48spaces$ The user could be AuthToken or the User/UniqueNick (with the extra PartnerID). server challenge that we received before. the client challenge that was generated before.

8.2.2 SDK Revision

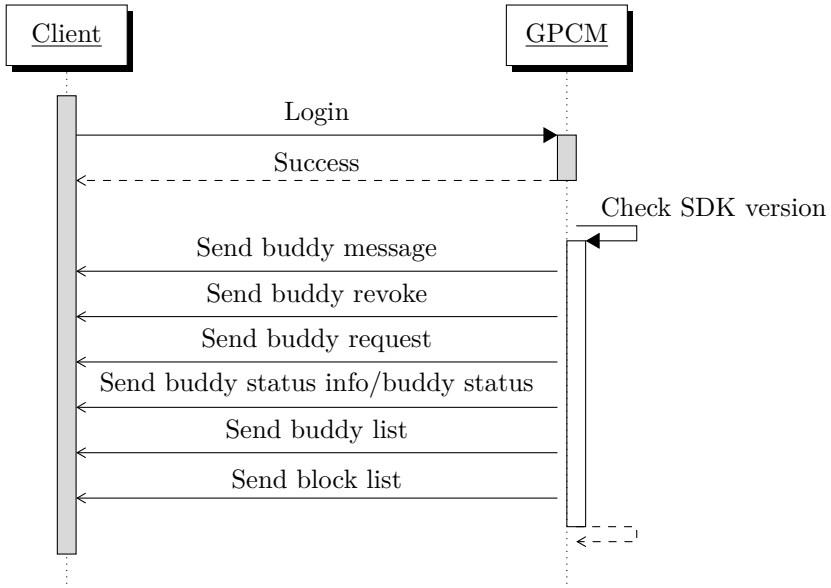


Figure 8.2: SDK Revision process

When a player finished login, GPCM will check his sdkrevision, sdkrevision is an addition of each sdkrevision number. Every addition of sdkrevision number will make GPCM act differently.

Code 8.2.6

- Extended message support
  - 1 GPI_NEW_AUTH_NOTIFICATION = 1
  - 2 GPI_NEW_REVOKE_NOTIFICATION = 2
- New Status Info support
  - 4 define GPI_NEW_STATUS_NOTIFICATION = 4
- Buddy List + Block List retrieval on login
  - 8 GPI_NEW_LIST_RETRIEVAL_ON_LOGIN = 8
- Remote Auth logins now return namespaceid/partnerid on login
  - 16 GPI_REMOTEAUTH_IDS_NOTIFICATION = 16
- New CD Key registration style as opposed to using product ids
  - 32 GPI_NEW_CDKEY_REGISTRATION = 32

For now, we know the sdkrevision number of GameSpy SDK test and Crysis2.

8.3 GPI Buddy Module

8.3.1 Buddy Message

The Buddy Message is a method to transmit message, buddy add request, game invite, friend revoke(friend deletion), buddy status(online status etc.).

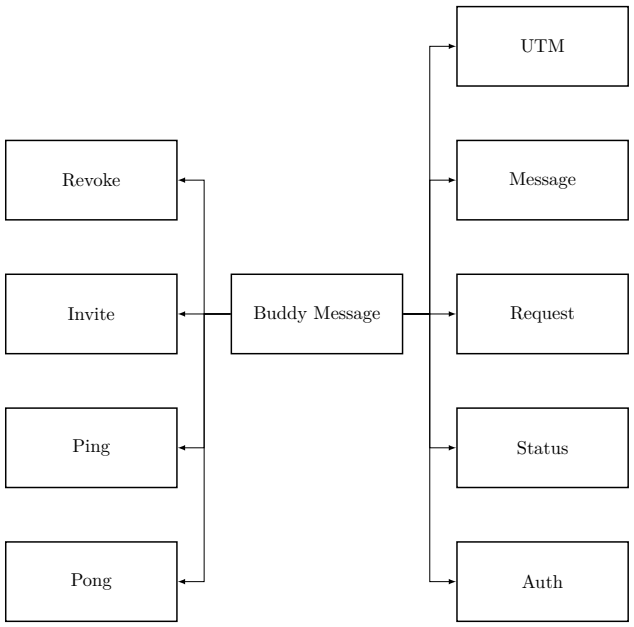


Figure 8.3: Buddy message module

When a Buddy Message received by a client, the client will determine Buddy Message type according to Table 8.5.

Definition	**Value**
GPI_BM_MESSAGE	1
GPI_BM_REQUEST	2
GPI_BM_REPLY	3
GPI_BM_AUTH	4
GPI_BM_UTM	5
GPI_BM_REVOKE	6
GPI_BM_STATUS	100
GPI_BM_INVITE	101
GPI_BM_PING	102
GPI_BM_PONG	103
GPI_BM_KEYS_REQUEST	104
GPI_BM_KEYS_REPLY	105
GPI_BM_FILE_SEND_REQUEST	200
GPI_BM_FILE_SEND_REPLY	201
GPI_BM_FILE_BEGIN	202
GPI_BM_FILE_END	203
GPI_BM_FILE_DATA	204
GPI_BM_FILE_SKIP	205
GPI_BM_FILE_TRANSFER_THROTTLE	206
GPI_BM_FILE_TRANSFER_CANCEL	207
GPI_BM_FILE_TRANSFER_KEEPALIVE	208

Table 8.5: Buddy Message Definition

Because Client1 and Client2 are in NAT network, so they can not connect each other using p2p, so GPCM will forward message for them. The forward diagram shows in Figure 8.4

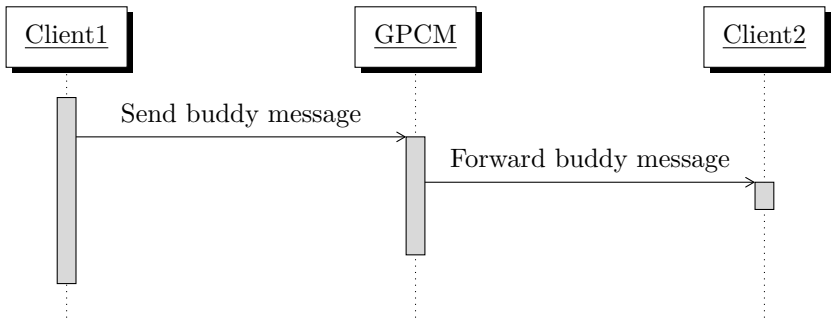


Figure 8.4: Buddy message diagram

All Buddy Message from Client will have same prefix which we show in 8.3.1

Client request:

Code 8.3.1

\bm\<buddy message type>\sesskey\<session key>
\t\<profile id>\date\<date>\$\cdots$\final\

Keys	Description	Type
bm	Indicate the buddy message command, please see 8.5	Uint
t	Profileid of the receiver	Uint
sesskey	The session key of the sender client	Uint
msg	The message contents	String

Table 8.6: Client buddy message command in prefix

All Buddy Message from GPCM will have same prefix which we show in 8.3.2. The contents in $\cdots$ is different from each Buddy Message Type.

Server response:

Code 8.3.2

\bm\<buddy message type>\f\<profile id>\date\<date>\$\cdots$\final\

Keys	Description	Type
bm	Indicate the buddy message command, please see 8.5	Uint
f	Profileid of the sender	Uint
date	The date that this message is sent, this value can be empty, possible format should be *xxxxxxxx* e.g. 20200201	Uint
msg	The message contents	String

Table 8.7: Buddy message command in prefix

Next following subsections we introduce message contents, the message content will use in both client buddy message and server buddy message. We only write the message contents after \msg\.

8.3.1.1 Message

This is a general message

Code 8.3.3

```
\msg\<message content>\final\
```

8.3.1.2 UTM

Code 8.3.4

```
\msg\<UTM message>\final\
```

8.3.1.3 Request

This is a add friend request.

Server response:

Code 8.3.5

```
\msg\|signed|<signature>\final\
```

8.3.1.4 Auth

Auth method is a add friend function. Auth method do not have contents after *date*\.

8.3.1.5 Revoke

Revoke method is called when a client1 deleted a client2 in his friend list. When deletion is finished in client1, client1 will send revoke message to GPCM, GPCM will forward this message to client2, then client2 will delete player1 in his friend list. Revoke method do not have contents after *date*\.

8.3.1.6 Status

This is an old method for game to get status information. buddy status 8.3.1.6 and buddy status info 8.3.2 can not be used at same time. Buddy status method is a part of Buddy Message module, old game send buddy status through a buddy message.

Server response:

Code 8.3.6

```
\msg\|s|<status code>|ss|<status string>|ls|<location string>
|ip|<ip address>|p|<port>|qm|<quiet mode flag>\final\
```

8.3.1.7 Invite

Invite method is used to invite a player to a game which is currently playing by another player.

Client request:

Code 8.3.7

```
\msg\|p|<product id>|l|<location string>\final\
```

8.3.1.8 PING

Ping method maybe is used to check the ping to other player.

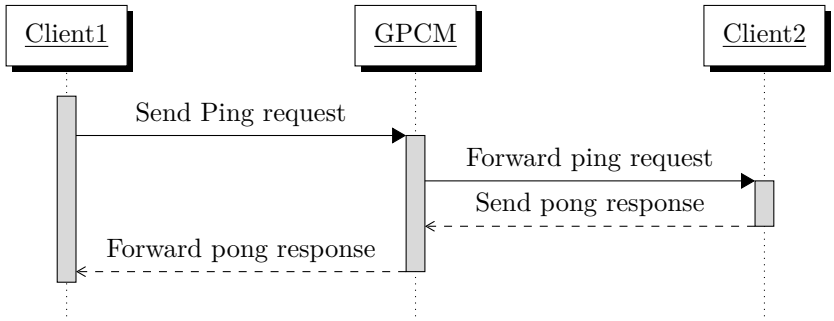


Figure 8.5: PING and PONG diagram

Client request:

Code 8.3.8

```
\msg\\final\
```

8.3.1.9 PONG

Server response:

Code 8.3.9

```
\msg\1\final\
```

8.3.2 Buddy Status Info

This is a new method used in new game. 8.3.1.6 is an old method used in old game. Currently we can not tell you which game use new method and which use old method.

Server response:

Code 8.3.10

```
\bsi\\state\<buddy status>\profile\<profileid>\bip\<buddy ip>
\bport\<buddy port>\hostip\<host ip>\hprivip\<host private ip>
\qport\<query port>\hport\<host port>\sessflags\<session flags>
\rstatus\<rich status>\gameType\<game type>\gameVnt\<game
variant>\gameMn\<game map name>\product\<productid>
\qmodeflags\<quiet mode flags>\final\
```

Keys	Description	Type
bsi	buddy status info command	
state	Buddy status state	Enum
profileid	The profileID	Uint
bip	Buddy ip	String
bport	Buddy port	Uint
hostip	Host ip	String
hprivip	Host private ip	String
qport	Query port	Uint
hport	Host port	Uint
sessflags	Session flag	Uint
rstatus	Rich status ?	String
gameType	Game type	String
gameVnt	Game variant	String
gameMn	Game map name	String
product	Productid	uint
qmodeflags	Quiet mode flag	Enum

Table 8.8: Buddy status info keys

8.3.3 Buddy List

Buddy list is a list which contains your friends. GPCM server will send buddy list when a client is logged in. Process is showing in Fig 8.6 and the response is showing in 8.3.3.

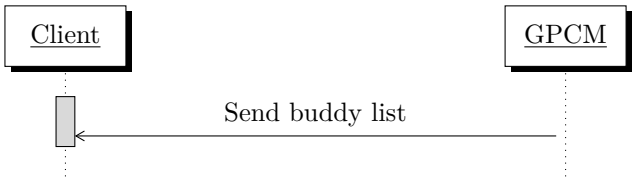


Figure 8.6: Buddy List

Server response:

Code 8.3.11

\bdy\<number of profileid>\list\<profileid 1>,
<profileid 2>,$\cdots$,<profileid n>\final\

8.3.4 Block List

Block list is an list which contain the players you do not like. GPCM server will send block list when a client is logged in. Process is showing in Fig 8.7 and the response is showing in 8.3.12.

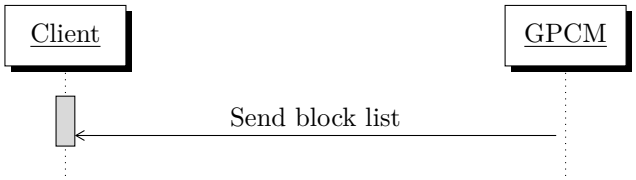


Figure 8.7: Block List

Server response:

Code 8.3.12

\blk\<number of profile id>\list\<profileid 1>,<profileid 2>,$\cdots$,
<profileid n>\final\

8.3.5 Add Buddy

When a client want to add another client into his buddy list. He will send the following request to GPCM.

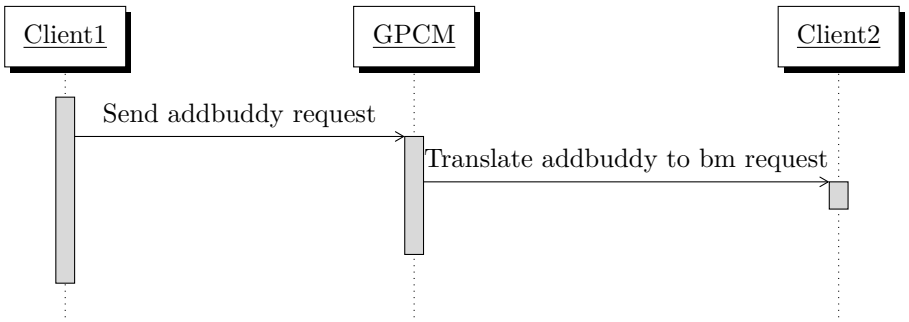


Figure 8.8: Add friend diagram

Client request:

Code 8.3.13

```
\addbuddy\\sesskey\<session key>\newprofileid\<profile id>
\reason\<add friend reason>\final\
```

8.3.6 Delete Buddy

When a client want to delete a friend in his buddy list. He will send the following request to GPCM.

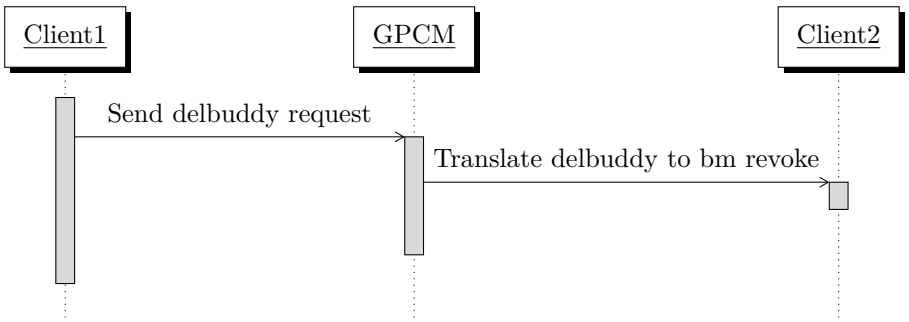


Figure 8.9: Delete friend diagram

Client request:

Code 8.3.14

```
\delbuddy\\sesskey\<session key>\delprofileid\<profile id>\final\
```

8.3.7 Add Block

Client request:

Code 8.3.15

```
\addblock\\sesskey\<session key>\profileid\<profile id>\final\
```

8.4 GPI Info Module

8.4.1 Profile

8.4.1.1 Get Profile Information

Find a user's profile information. signature string in response is used in adding someone as your friend through buddy message.

Client request:

Code 8.4.1

```
\getprofile\\sesskey\<session key> \profileid\<profile id>
\id\<operation id>\final\
```

Server response:

Code 8.4.2

```
\pi\\profileid\<profile id>\nick\<nick name>
\uniquenick\<uniquenick>\email\<email>\firstname\<first name>
\lastname\<last name>\icquin\<icquin>
\homepage\<home page URL>\zipcode\<zip code>
\countrycode\<country code>\lon\<longitude>\lat\<latitude>
\loc\<location>\birthday\<birthday>\sex\<gender>
\pmask\<public mask>\aim\<aim name>\pic\<picture>
\occ\<occupation id>\ind\<industry id>\inc\<income id>
\mar\<married id>\chc\<child count number>\i1\<interest 1>
\o1\<ownership 1>\conn\<connection type id>
\sig\<peer to peer signature>\id\<operation id>\final\
```

Keys in profile module:

Key	Description
cpubrandid	cpu barand id
cpuspeed	cpu speed
memory	memory
videocard1ram	GPU memory size
videocard2ram	GPU memory size
connectionid	connection id
connectionspeed	connection speed
hasnetwork	unknow
passwordenc	encrypted password

Table 8.9: Other keys in profile

8.4.1.2 Update Profile Information

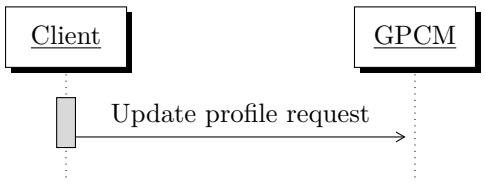


Figure 8.10: Update profile diagram

Client request:

Code 8.4.3

```
\updatepro\\sesskey\<session key>\⋆\partnerid\<partner id>\final\
```

The ⋆ in 8.4.3 is the profile information key and value pairs such as **\nick\<nick name>**, etc.

8.4.1.3 Update User Information

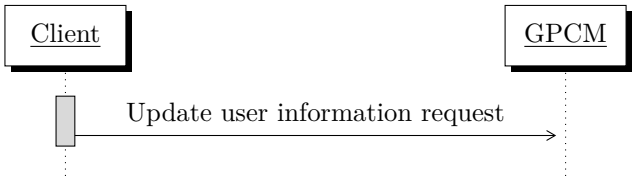


Figure 8.11: Update user information diagram

Client request:

The ⋆ in **??** is the profile information key and value pairs such as **\passwordenc\<encrypted password>**, etc.

Code 8.4.4

```
\updateui\\sesskey\<session key>\*\final\
```

8.4.2 GPI Profile Module

8.4.2.1 Create New Profile

Create a new profile with nick name.

Client request:

Code 8.4.5

```
\newprofile\\sesskey\<session key>\nick\<nick name>\id\<operation
id>\final\
```

8.4.2.2 Replace Existed Profile

Replace nick name in a profile with a new nick name.

Client request:

Code 8.4.6

```
\newprofile\\sesskey\<session key>\nick\<old nick name>\replace\1
\oldnick\<nick name>\id\<operation id>\final\
```

8.4.2.3 Delete Profile

Client request:

Code 8.4.7

```
\delprofile\\sesskey\<session key>\id\<operation id>\final\
```

Server response:

Code 8.4.8

```
\dpr\final\
```

8.4.3 GPI Unique Module

8.4.3.1 Register Unique Nick

This method will register a new unique nick. There are two request 8.4.9 and 8.4.10. The first one is only register unique nick, and the second one is register unique nick with cd key.

Client request:

Code 8.4.9

```
\registernick\\sesskey\<session key>\uniquenick\<unique nick>
\partnerid\<partner id>\id\<operation id>\final\
```

Code 8.4.10

```
\registernick\\sesskey\<session key>\uniquenick\<unique nick>
\cdkey\<cd key>\partnerid\<partner id>\id\<operation id>\final\
```

Server response:

Code 8.4.11

```
\rn\final\
```

8.4.3.2 Register CD Key

Client request:

Code 8.4.12

```
\registercdkey\\sesskey\<session key>\cdkeyenc\<cd key enc
string>\id\<operation id>\final\
```

Server response:

Code 8.4.13

```
\rc\final\
```

8.4.4 GPI Peer Module

8.4.5 GPI Transfer Module

Chapter 9

GameSpy Presence Search Player

GPSP server provides search function for client.
Table 7.1 are the GPSP IP and Ports that client/game connect to.

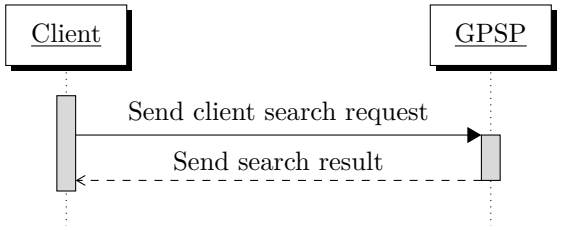


Figure 9.1: GPSP diagram

9.1 Search Profile

Client request:

Code 9.1.1

\search\\sesskey\<session key>\profileid\<profile id>\⋆
\namespaceid\<namespace id>\partnerid\<partner id>
\gamename\<game name>\final\

Symbol ⋆ contains client detail, we list client detail as follows.

Code 9.1.2

\nick\<nick name>\uniquenick\<unique nick>\email\<email>
\firstname\<first name>\lastname\<last name>\icquin\<icq uin>
\skip\<skip>

Server response:

Code 9.1.3

\bsr**<profile 1>**\bsr**<profile 2>**\bsr\$\cdots$**<profile n>**
\bsrdone\\more\<number of rest profiles>\final\

The value in **<profile i>** is showing below 9.1.4.

Code 9.1.4

<profileid>\nick\<nick>\uniquenick\<unique nick>
\namepaceid\<namespace id>\firstname\<first name>
\lastname\<last name>\email\<email>

9.1.1 Seach Profile With Unique Nick

Client request:

Code 9.1.5

\searchunique\\sesskey\<session key>\profileid\<profile id>
\uniquenick\<unique nick>\namespaces\<namespace id 1,
namespace id 2, ..., namespace id n>\final\

Server response:

The response from server is the same as 9.1.3.

9.1.2 Search User Is Valid

Client request:

Code 9.1.6

\valid\\email\<email>\partnerid\<partner id>\final\

Server response:

Number 0 represents false, 1 represents true.

Code 9.1.7

\vr\<valid code: 0 or 1>\final \

9.1.3 Search Nick

This method is used to search profile with nick name and email.

Client request:

Code 9.1.8

```
\nicks\\email\<email>\passenc\<encrypted password>
\namespaceid\<namespace id>\partnerid\<partner id>
\gamename\<game name>\final\
```

Server response:

Code 9.1.9

```
\nr\\nick\<data 1>\<data 2>\···\<data n> \ndone\final\
```

The content in **<data i>** shows below.

Code 9.1.10

```
<nick name>\uniquenick \<unique nick>
```

9.1.4 Search Player

Client request:

Code 9.1.11

```
\pmatch\\sesskey\<session key>\profileid\<profile id>
\productid\<product id>\gamename\<game name>\final\
```

Server response:

Code 9.1.12

```
\psr\<data 1>\psr\<data 2>\···\psr\<data n>\psrdone\final\
```

The content in **<data i>** shows below.

Code 9.1.13

```
<profile id>\status\<status string>\nick\<nick name>
\statuscode\<status code>
```

9.1.5 Search Check

This method is used to check whether user exist.

Client request:

Code 9.1.14

```
\check\\nick\<nick name>\email\<email>\partnerid\<partner id>
\passenc\<encrypted password>\gamename\<game name>\final\
```

Server response:

The error code in 9.1.15 shows in .

Code 9.1.15

```
\cur\<check error code>\pid\<profile id>\final\
```

9.1.6 User Creation

This commmand 9.1.16 is used to create a user in GameSpy.

Client request:

Code 9.1.16

```
\newuser\email \<email>\nick\< nick name>
\passwordenc\<password enc>\productid\<product id>
\uniquenick\<unique nick> \cdkeyenc\<cdkeyenc>
\partnerid\<partnerid>\gamename\<gamename>\final\
```

Server response:

The newuser error code shows in .

Code 9.1.17

```
\nur\<newuser error code>\pid\<profile id>\final\
```

9.1.7 Search Others Buddy

Client request:

Code 9.1.18

```
\others\\sesskey\<session key>\profileid\<profile id>
\namespaceid\<namespace id>\gamename\<game name>\final\
```

Server response:

GPSP should try to find the information, if some account do not have unique nick then do not add \uniquenick\<unique nick>\to response string.

Code 9.1.19

\others\\o\<data 1>\o\<data 2>\$\cdots$\\<data n>\odone\final\

The content in **<data i>** is listed as follows.

Code 9.1.20

<profile id>\nick\<nick name>\uniquenick\<unique nick>
\first\<first name>\last\<last name>\email\<email>

9.1.8 Search Others Buddy List

Client send request to GPSP asking for the buddy's profiles with buddy profile id.

Client request:

Code 9.1.21

\otherslist\\sesskey\<session key>\profileid\<profile id>
\numopids\<number of recieved buddy profiles>
\opids\<profile id 1>|<profile id 2>|$\cdots$|<profile id 3>
\namespaceid\<namespace id>\gamename\<game name>\final\

Server response:

Code 9.1.22

\otherslist\\o\<data 1>\o\<data 2>\$\cdots$\\<data n>\odone\final\

The content in **<data i>** is listed as follows.

Code 9.1.23

<profile id>\uniquenick\<unique nick>

9.1.9 Search Suggest Unique

Client search suggest nick name on GPSP.

Client request:

Code 9.1.24

\uniquesearch\\preferrednick\<unique nick name>
\namespaceid\<namespace id>\gamename\<game name>\final\

Server response:

Code 9.1.25

\us\<number of suggest nick>\nick\<nick name1>
\nick\<nick name2>\$\cdots$\nick\<nick name n>\usdone\final\

9.1.10 Valid Email

Client request:

Code 9.1.26

\valid\\email\<email account>\partnerid\<partner id>
\gamename\<game name>\final\

Server response:

Code 9.1.27

\vr\<valid value>\final\

Part IV

Transport

Part V

NAT Negotation

Chapter 10

Introduction

The GameSpy NAT Negotiation SDK interacts with GameSpy's NAT Negotiation server to allow hosting of multiplayer games by users behind NAT and firewall devices. Typically, a user behind a NAT or firewall device cannot host multiplayer games because the device will block incoming connections from outside users. GameSpy's NAT Negotiation technology allows two users, one or both of whom are behind a NAT device, to open a clear UDP channel directly between the users. GameSpy's NAT Negotiation technology uses a method known as "Port Guessing" to attempt to discern future port mapping information for two users based on their connections to the NAT Negotiation server. Once this mapping information is determined, the server exchanges the information with the users, and they connect to each other directly (note: the term "connect" in this document is understood to mean the establishment a clear, two-way channel between the users, since UDP is in reality a connection-less protocol).

Note that the NAT Negotiation SDK does not make any distinction between the "client" who is connecting to a "server" (or "host"), however this document will use those terms for clarity, and because the other SDKs involved do make that distinction.

The NAT Negotiation SDK itself is very simple - two users who want to be connected to each other have a shared "cookie" value that the NAT Negotiation server uses to match the users up.

The NAT Negotiation SDK has no limit to the number of users that can be connected together, but each channel between two users must be independently established.

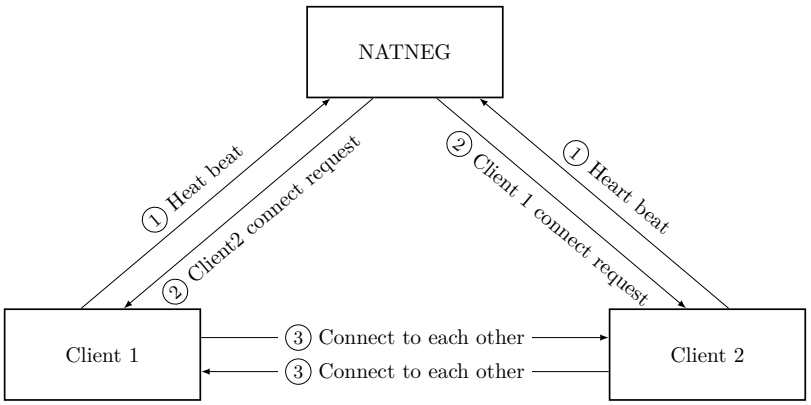


Figure 10.1: NatNeg working diagram

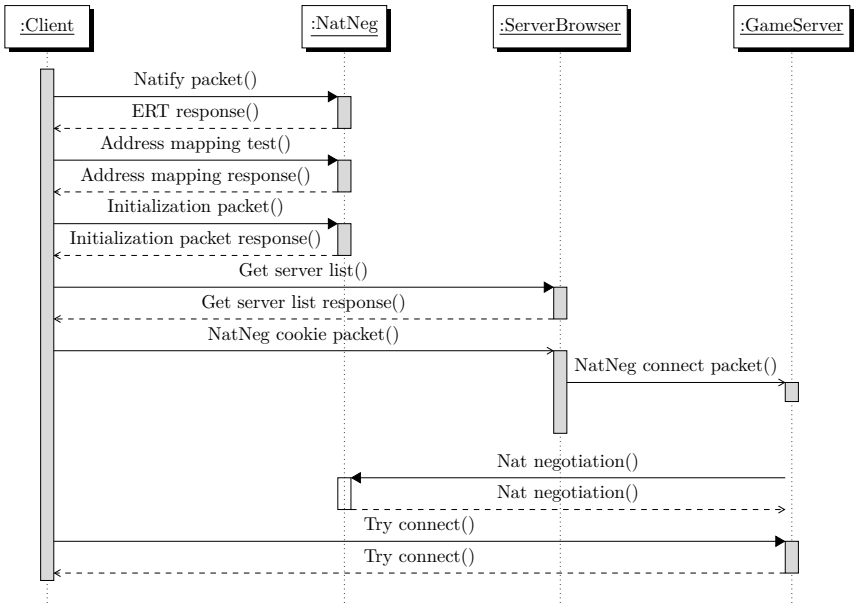


Figure 10.2: NatNeg sequence diagram

Name	IP	Port
NATNEG	natneg1.gamespy.com	27901 (udp)
NATNEG	natneg3.gamespy.com	27901 (udp)
NATNEG	natneg2.gamespy.com	27901 (udp)

Table 10.1: IP and Ports for NatNeg Servers

Nat Negotiation mechanism: Because the ip address and other environment are changing from time to time, so when a client1 wants to connect to client2, he dose not know any informations about client2, so he cannot connect to client2.

using natneg it can ask client2 information on gamespy nat server and connect to client2.

Nat Negotiation SDK do the following things:

- Clients connect to GameSpy NatNeg server
- Clients send the heart beat data that contain all information about himself to GameSpy NatNeg server
- GameSpy Nat server store clients information.
- when a client1 is try to connect to other client2:
  - client1 send request to GameSpy NatNeg server
  - GameSpy NatNeg server send the information about client2 to client1
  - client1 get the client2 information and connect.

10.1 NetNag Packet

Code 10.1.1

```
typedef struct _NatNegPacket
{
unsigned char magic[NATNEG_MAGIC_LEN];
unsigned char version;
unsigned char packettype;
int cookie;

union
{
InitPacket Init;
ConnectPacket Connect;
ReportPacket Report;
} Packet;

} NatNegPacket;
```

10.1.1 Magic Data

Every heart beat packet start with magic data.

Code 10.1.2

```
Magic Data: 0xFD 0xFC 0x1E 0x66 0x6A 0xB2
```

10.1.2 NatNeg Packet Type

Client's heart beat contains NatNeg packet type which we list as follows.

Packet type	Description	Value
Init		0
ErtTest	Echo packet to original sender test	2
Connect		5
Connect Ack		6
Connect ping		7
BackupTest		8
Address check		10
Natify request	NAT identify	12
Report	NatNeg result report	13

Table 10.2: NatNeg client request packet type

Packet type	Description	Value
InitAck		1
ErtAck	External reach test	3
Backup Ack		9
Address reply		11
Report Ack		15

Table 10.3: NatNeg server response packet type

10.1.3 Initial Packet

Code 10.1.3

```
typedef struct _InitPacket
{
unsigned char porttype;
unsigned char clientindex;
unsigned char usegameport;
unsigned int localip;
unsigned short localport;
char[] gamename;
} InitPacket;
```

10.1.4 Report Packet

Code 10.1.4

```
#define REPORTPACKET_SIZE BASEPACKET_SIZE + 61
typedef struct _ReportPacket
{
unsigned char porttype;
unsigned char clientindex;
unsigned char negResult;
NatType natType;
NatMappingScheme natMappingScheme;
char gamename[50];
} ReportPacket;
```

10.1.5 Connect Packet

Code 10.1.5

```
#define CONNECTPACKET_SIZE BASEPACKET_SIZE + 8
typedef struct _ConnectPacket
{
unsigned int remoteIP;
unsigned short remotePort;
unsigned char gotyourdata;
unsigned char finished;
} ConnectPacket;
```

10.2 Nat Negotiation Process

Natify –> AddressCheck –> Init –> Connect –> ConnectPing –> Report

10.2.1 Nat Identification

When client start, it sends 3 different Natify packet (NN1,NN2,NN3)to NatNeg server to discover it's reach-ability.

ERT stands for external reach test, which detect network environment. check your private ip and port whether equal to your public ip and port.

10.2.2 Address Check

Then client will send 4 address check packets to NatNeg server to discover network mapping. Each packet contains cookie defined before which are {packet_map1a, packet_map1b, packet_map2, packet_map3}.

10.2.2.1 Initial NatNeg

Client request:

Client sends a initpacket which contains an extra information of gamename to NatNeg server.

Server response:

Server changes the packet type of received initpacket to InitAck then send back this packet to sender.

Part VI

Peer to Peer communication

Chapter 11

Peer to Query Report Server

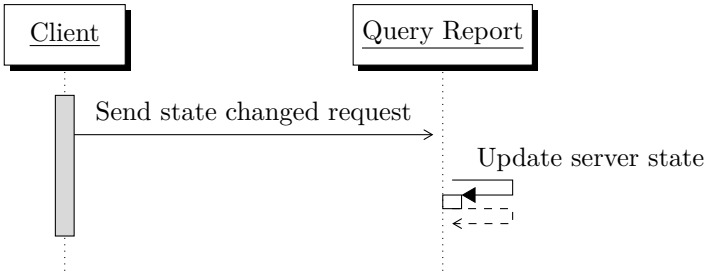


Figure 11.1: Peer to query report diagram

Chapter 12

Peer to Server Browser Server

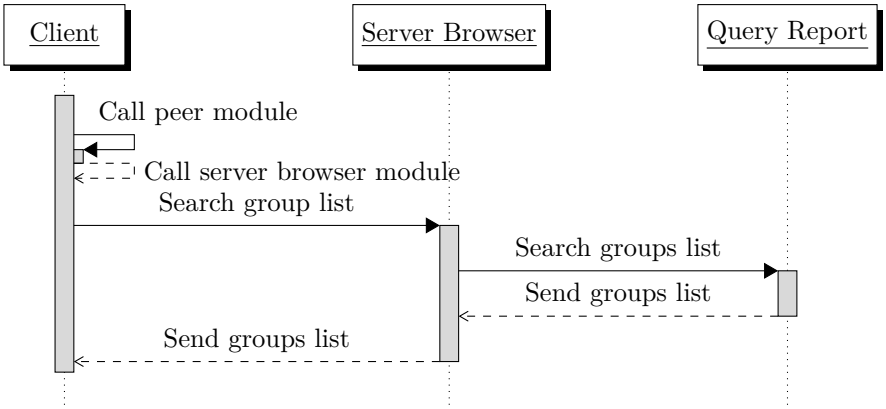


Figure 12.1: Peer to server browser diagram

Part VII

Patching & Tracking

Part VIII

Query & Reporting

Query Report server is responsible for server available check and dedicated server information collection. Query Report server and Server Browser server communicate with each other, these two servers generally called master server.

Name	IP	Port
QR	*.master.gamespy.com	27900 (Udp)

Table 12.1: IP and Ports for Query Report Server

Chapter 13

Avaliable Check

According to GameSpy SDK every game needs to check GameSpy back end server before using multiplayer. The contents in client request are shown below.

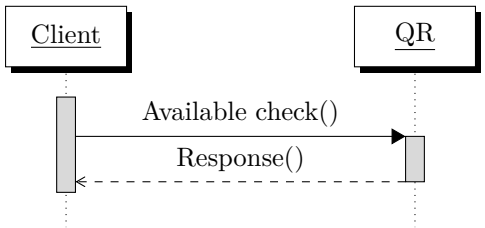


Figure 13.1: The sequence of available check

Client request:

Code 13.0.1

```
PacketType:0x09
GameName:<game name>
```

The response of available check is fixed with first 6 bytes, last byte is the status of the master server.

Status	Description	Value
GSIACWaiting	still waiting for a response from the backend	1
GSIACAvailable	the game's backend services are available	2
GSIACUnavailable	the game's backend services are unavailable	3
GSIACTemporarilyUnavailable	the game's backend services are temporarily unavailable	4

Table 13.1: Available check server status

Server response:

Code 13.0.2

0xfe, 0xfd, 0x09, 0x00, 0x00, 0x00, <server status 13.1>

Chapter 14

Game Server Information Report

Query Report server collecting information about the player, server, team.

14.1 General Information

Data type	Description
Server data	This is general information about the game in progress, for example - the map that is being played, the type of game, and any specific game settings that would be of interest to players before they joined.
Player data	This is information about a specific player that is in the current game, for example - the player's name, their current score, what team they are on, and the latency to the game server.
Team data	This is information about a specific team in the current game, for example - the name of the team and the team score. If your game does not support team play you do not need to report any team information.

Table 14.1: The report data type

The format of the data in report data is key value pair. GameSpy gives several GameSpy defined keys that we list as follows and game can define their custom key to report.

Key	Description
hostname	a descriptive host-defined string (can include spaces) that identifies the server (e.g. "Joe's Game!")
gamever	a version specifier (e.g. 1.23)
hostport	the port that the game networking is running on and that the client should connect to. If the game shares a port with the Query and Reporting 2 SDK, you do not need to specify this.
mapname	the map name (either filename or descriptive name)
gametype	string which specifies the type of game, or the mod being played.
gamevariant	if the particular game type has multiple variants, you can report it using this key.
numplayers	numeric string, number of players on the server
numteams	numeric string, number of teams on the server
maxplayers	numeric string, max number of players for this server
gamemode	string which specifies what is going on in the game at that time. see Table 14.3
teamplay	number which defines the type of teamplay in use, or 0 for no teamplay. Values > 0 are up to the developer
fraglimit	number of total kills or points before a level change or game restart
teamfraglimit	number of total kills or points for a team before a level change or game restart
timelimit	amount of total time before a level change or game restart occurs (generally in minutes)
timeelapsed	amount of time (in seconds) since the current level or game started
roundtime	amount of time before a round ends (for round based games)
roundelapsed	amount of time (in seconds) that the current round has been in progress
password	0 or not present if no password is required to join, 1 if password is required. Implementation of actual password protection is up to the game developer's network code.
groupid	(optional) If the server being hosted is part of a "group room" then it needs to report which groupid it is part of (as passed in on launch)
player_	a player name (may include spaces)
score_	numeric string that contains the score (kills/points) for a single player
skill_	a skill rating, if applicable, for a single player
ping_	the ping for a player (as measured between the player and the server)
team_	the team a player is on, either numeric or string
deaths_	number of deaths a player has had
pid_	The profileID number for a player (if logged in with the P&M SDK)
team_t	the name for a team
score_t	the score for a team

Table 14.2: Report data keys

Packet type	Value
Query	0x00
Echo response	0x05
Client Message	0x06
Message ACK	0x07
Keep alive	0x08

Table 14.4: Query report client packet

Packet type	Value
Pre-query ip verify	0x09
Challenge	0x01
Echo	0x02
AddError	0x04
Keep alive	0x08

Table 14.5: Query report server packet

Key	Description
openwaiting	game has not yet started and players can join
closedwaiting	game has not yet started and players cannot join
closedplaying	game is in progress, no joining allowed
openplaying	game is in progress, players may still join
openstaging / closedstaging	Use to report that the game is in staging mode (should generally not be used directly - the Peer SDK handles this automatically).
exiting	server is shutting down

Table 14.3: Game mode detail

14.2 Pre-Query IP Verify Packet

- Server uses this packet to verify the IP of a client that connects to the server.
- This is only done if the user enable IP challenge.
- Each new client has to verify them-self with a challenge.

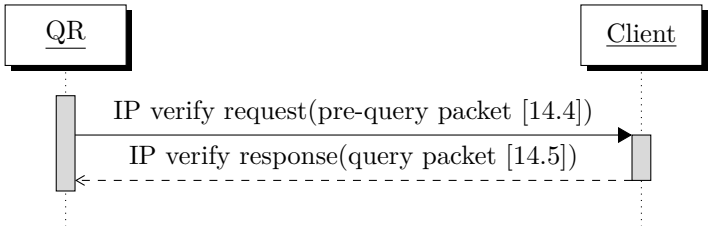


Figure 14.1: The sequence of IP verify

This packet has limit length 200 bytes.

Server response:

Code 14.2.1

|Magic data(2 bytes)|Packet type (1 byte)|Instant key(4 bytes)|Server challenge (int)|

14.3 Query Packet

After receiving the pre-query ip verify request from server, client will compute the challenge response and send to server.

14.4 Heart Beat Packet

Heat beat packet is used to report the information about server, player and team. We guess when QR server received a heartbeat packet, it will generate an challenge and send to game server. after received challenge game server will compute response and send to QR server to verify. If game server response is valid then QR server will add this server to server list.

Custom keys are used to define custom data to report, for example if the user is playing with a Windows or Macintosh.

There could be two types of custom keys: Player keys (they end with _): Custom player information Team keys (they end with _t): Custom team (or brigade) information Server keys (they don't end with anything): Custom server information

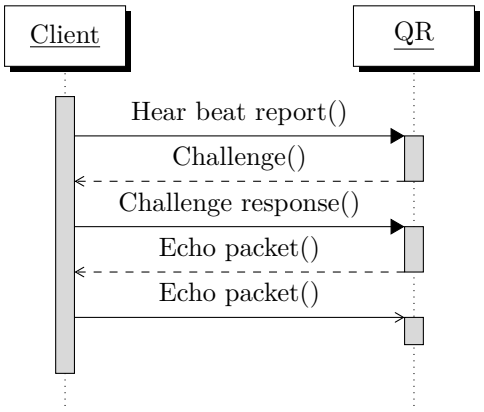


Figure 14.2: The sequence of heart beat packet process

Client request:

Code 14.4.1

|Instant key (4 bytes) | \0\0 |Total number of server keys (1 byte) | Server keys (unknown bytes) | \0\0 | Total number of player keys (1 byte) | Player keys (unknown bytes) | \0\0 |Total number of team keys (1 byte) | Team key (unknown byte) |

14.5 Challenge Packet

When received the heart beat packet from client, Query Report server sends a challenge packet to verify the authenticity of the report client to prevent some illegal client.

Server response:

Code 14.5.1

| Magic data | Packet type | Instant key | Challenge (5 bytes) | User IP (4 byte) | 4 bytes padding | User port (4 byte) |

14.6 Echo Packet

We guess echo packet is used to detect the ping between server and Query Report server.

14.7 AddError Packet

This packet means address error, when query report server find client ip is not valid, server will send this packet to client.

Chapter 15

The Process of CD key or Nat Negotiation authentication with Query Report

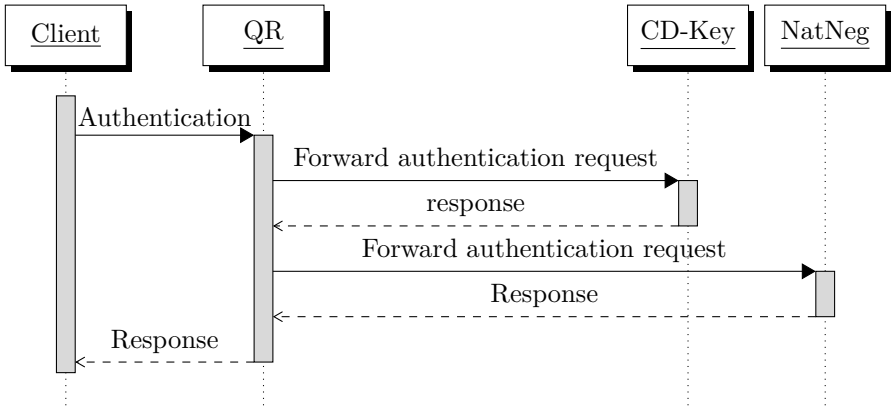


Figure 15.1: The diagram of CD key or NatNeg server authentication with query report

Part IX

Server Browser

Chapter 16

Overview

The GameSpy Server Browsing SDK is a portable LAN and Internet server browser engine. It allows developers to quickly and easily add a list-based matchmaking interface to the game, with powerful features such as server-side filtering, sorting, country-filtering, and ping (latency) measurement.

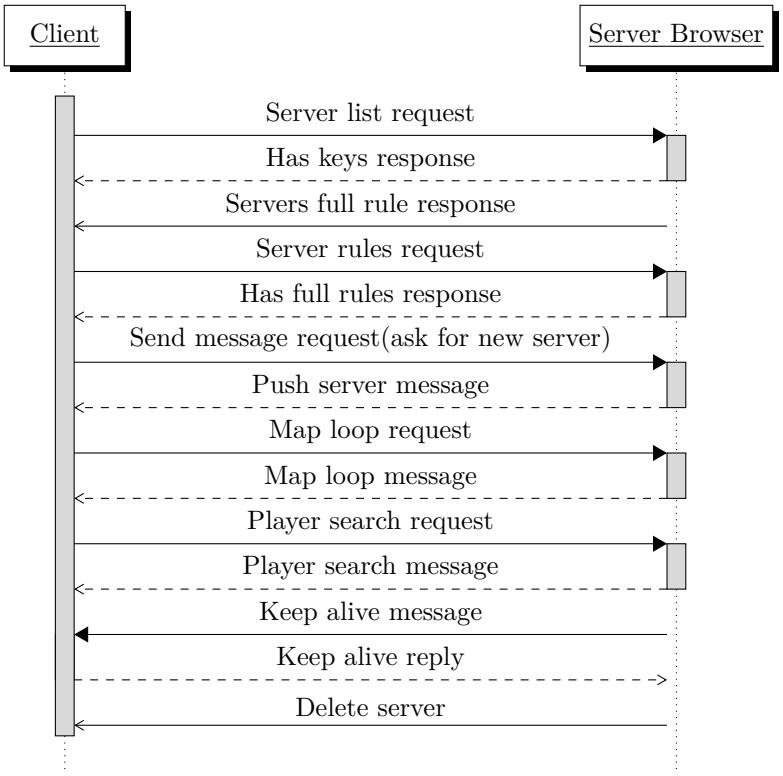
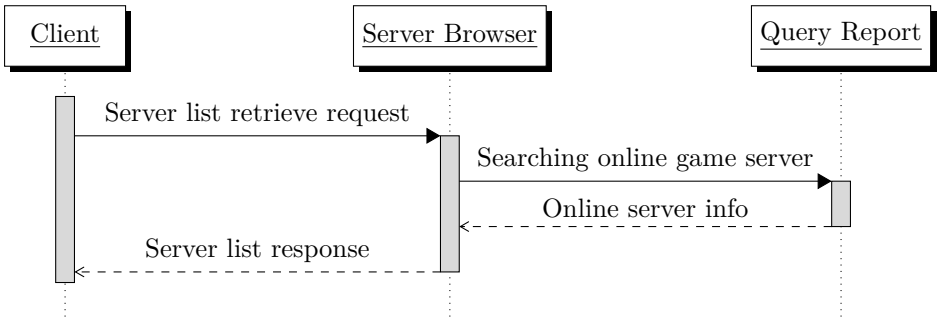


Figure 16.1: Total process of server browser server

Chapter 17

Server List Retrieve

When a client want to see how many servers are online for playing it will do the following.



Note: The port in request and response should be check if it is little Indian, and according to the result we should do byte reverse on port.
NTS string means Null Terminated String which ends with \0.

Client request:

Code 17.0.1

```
Total length of the request (2 bytes)
Protocol version (1 byte)
List protocol version (1 byte)
Encoding version (1 byte)
Game version (4 bytes)
Game name for development(i bytes)
Game name for real application(i bytes)
Challenge (8 bytes)
Server filter(i bytes)
Key field list(i bytes)
Query option (4byte)
Source IP (4 bytes) [optional]
Max servers (4 bytes) [optional]
```

After query option there could be more bytes which is listed as follows. If there has alternate source ip option the following should be added after query option. If there has limit result count option the following should be added after query option.

Name	Value
SERVER_LIST_REQUEST	0
SERVER_INFO_REQUEST	1
SEND_MESSAGE_REQUEST	2
KEEPALIVE_REPLY	3
MAPLOOP_REQUEST	4
PLAYERSEARCH_REQUEST	5

Table 17.1: Server browser request

Name	Value
PUSH_KEYS_MESSAGE	1
PUSH_SERVER_MESSAGE	2
KEEPALIVE_MESSAGE	3
DELETE_SERVER_MESSAGE	4
MAPLOOP_MESSAGE	5
PLAYERSEARCH_MESSAGE	6

Table 17.2: Server browser response

The list protocol version=1 and Encoding version=3 in code 17.0.1. We do not know if there is another protocol version exist only by looking at the GameSpy SDK. Server filter can select the server according to condition client send to server browser. Filed list contains the keys that client need such as game name, game type, map name etc.

Update option name	Description	Value
SEND_FIELDS_FOR_ALL	Unknown	1
NO_SERVER_LIST	Only sends servers info do not send keys values etc.	2
PUSH_UPDATES	Updates server information in group list (used by peer)	4
ALTERNATE_SOURCE_IP		8
SEND_GROUPS	Sends groups information (used by peer)	32
NO_LIST_CACHE	Unknown	64
LIMIT_RESULT_COUNT	Sends servers with a limit number	128

Table 17.3: Server browser server list update options

Server response:

Code 17.0.2

Random byte (1 byte) ∧ 0xEC
Response Message Length ∧ xEA (1 byte)
Server Challenge (i bytes)
Client's public ip (4 bytes)
Standard query port (2 bytes)
Number of keys (1 byte)
Keys list → 17.1
Unique values list → 17.2
Servers info list → 17.3
AdHoc data → 17.4

17.1 Keys List

The key list in above contains all keys that client required for. Each key is separate by **0x00** and the first byte of each key is the key type. There are 3 types we listed as follow.

Key type	Value
Byte	0
Short	1
String	2

Table 17.4: The key type enumerator

Code 17.1.1

Key-1 type (1 byte), Key-1 name (i bytes)
Key-2 type (1 byte), Key-2 name (i bytes)
...
Key-n type (1 byte), Key-n name (i bytes)

17.2 Unique Values List

There are 2 ways to parse the value:

- Popular value: presents all string value inside unique value list, gives indexes in servers info list.
- Null terminate string: presents all string value inside server info list, sets the unique value number to 0.

Each unique value is separate by **0x00**.

Code 17.2.1

```
Unique value1 (i bytes)
Unique value2 (i bytes)
...
Unique value3 (i bytes)
```

17.3 Servers Info List

Servers info list contains multiple information of servers, each server is separated by server flags17.5, final server should end with **0x00FFFFFFFF**.

Code 17.3.1

```
Server flag (1 byte)
Server public ip (4 bytes)
Private port (4 bytes) [optional]
Private IP (4 bytes) [optional]
ICMP IP (4 bytes) [optional]
String index or NTS string
```

If you use popular value to parse values there should be string index which starts with 0. Each index represents the value index.

Code 17.3.2

```
0 | 1 | 2 | ··· | n
```

If you use NTS string you should parse value in this way:

Code 17.3.3

```
0xFF | value | 0x00
```

The game server flag in 17.0.2 is listed as follows.

Flag name	**Value**
UNSOLICITED_UDP_FLAG	1
PRIVATE_IP_FLAG	2
CONNECT_NEGOTIATE_FLAG	4
ICMP_IP_FLAG	8
NONSTANDARD_PORT_FLAG	16
NONSTANDARD_PRIVATE_PORT_FLAG	32
HAS_KEYS_FLAG	64
HAS_FULL_RULES_FLAG	128

Table 17.5: Server flags in servers info list

17.4 AdHoc Data

AdHoc data is used to help peer module setup game service.
If game want to use this function, client should require servers list 17 then the crypt header will be initialized which can be used to encrypt further communication.
AdHoc data contains the following types of command.

17.4.1 Push Keys List

Server response:

Code 17.4.1

Number of keys (1 byte)
Keys list with key type (i bytes) [NTS string]

17.4.2 Push Server

Search a server's keys and full rules from server browser.

Server response:

Code 17.4.2

Server flag (1 byte) 17.5
Server public IP (4 bytes)
Server public port (2 bytes)

17.4.3 Keep Alive

17.4.4 Delete Server

Delete a specific server in servers list.

Server response:

Code 17.4.3

Server public IP (4 bytes)
Server public port (2 bytes)

17.4.5 Map Loop

Gets a servers map information.

Client request:

Code 17.4.4

```
Length of request (2 bytes)
MAPLOOP_REQUEST (1byte)
Server public IP (4 bytes)
Server public port (2 bytes)
```

Server response:

Code 17.4.5

```
Server public IP (4 bytes)
Server public port (2 bytes)
Map change time (4 bytes)
Number of maps (1 byte)
Map names (i bytes) [NTS string]
```

17.4.6 Player Search

Search a specific player information.

Client request:

Total request length should less than 256 bytes.

Code 17.4.6

```
Length of request (2 bytes)
PLAYERSEARCH_REQUEST (1 byte)
Search options (4 bytes)
Max results (4 bytes)
Player name (i bytes)
```

Server response:

Player names are NTS string.

Code 17.4.7

```
Is final flag (1 byte)
Result count (1 byte)
Player names (i bytes)
Server IP (4 bytes)
Server ports (2 bytes)
Last record UTC time (4bytes)
Game name (i bytes)
```

Chapter 18

Server Info

Retrieves server rules for a specific server.

Client request:

Code 18.0.1

```
Length of request message (2 bytes)
SERVER_INFO_REQUEST (1 byte)
Public IP of specific server (4 bytes)
Public port of specific server (2 bytes)
```

Server response:

Code 18.0.2

```
```

Part X

SAKE Persistent Storage

Part XI

ATLAS Competition

Part XII

Voice Chat

Part XIII

Web Authentication

Part XIV

GameSpy Status & Tracking

Chapter 19

General Introduction

19.1 Note

Game uses GSTATS to store its data only using email and passwords login method in GPCM. So we do not need to consider namespaceid, we only need to find profileid.

19.2 Working Process

1. On startup, the host connects to tracking server, is authenticated, and is assigned a unique connection ID. If disk logging is enabled (see below) and there are logged games, they are sent to the tracking server.
2. When the actual game starts, the host sends a new game notification to the tracking server and creates internal structures for managing the game information.
3. During the game the host collects information into buckets (or developer's own data structures) and sends out snapshots at regular intervals (in case the host is reset before the game finishes)
4. (If player authentication is used) As players connect, the host sends out a challenge to the client, which formats a response based on its password or CD Key. This response is sent back to the host and stored as part of the snapshot.
5. When the game is complete, a final snapshot is sent to the tracking server.
6. A new game can be started immediately over the same connection (multiple simultaneous games over the same tracking server connection are supported as well).
7. The tracking server post-processes the data to extract some standard information and verify the authentication of the players. Disk logged or unusual games are marked for inspection.

19.3 Message Encryption

The GameSpy Stats & Tracking (GSTATS) SDK provides a simple, secure way to report the results and statistics of games to a central server. These results can then be used to help facilitate online rankings, ladders, and tournaments. Tracking is done in a very abstract manner than can be applied to any type of multiplayer game.
The communication between client and GSTATS server is encrypted under an simple XOR method. After encrypted the message look like 19.3.1.

Code 19.3.1

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\final\
```

19.4 Client Command

Command	Description
auth [20.1]	Authentication request
updgame	Sends a snapshot of information about the current game.
authp	Authenticates player
getpd	Gets persist data values modified
getpid	Ge sprofile id
newgame	Creates a new game for logging and registers it with the stats server.

Table 19.1: GSTATS client request command

19.5 Server IP and Port

Name	IP	Port
GSTATS	gamestats.gamespy.com	29920 (tcp)

Table 19.2: IP and Ports for GameSpy status and tracking

Chapter 20

Protocol Detail

20.1 Authentication

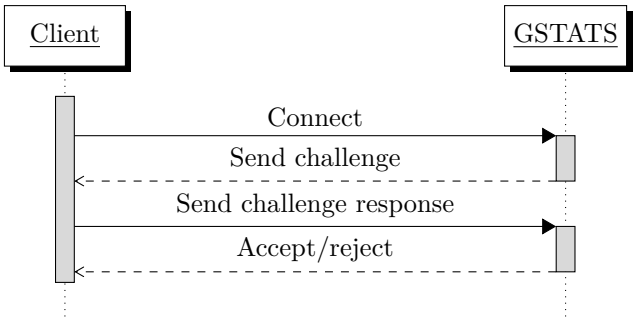


Figure 20.1: GSTATS authentication diagram

Server challenge:

Code 20.1.1

```
\challenge\<challenge string>\final\
```

Client request:

Code 20.1.2

```
\auth\\gamename\<game name>\response\<response string>
\port\<game port>\id\<operation id>\final\
```

The response string in20.1.2 is computed using the secrete key D of the client.

Server response:

Code 20.1.3

\sesskey\<session key>\final\

Session key should be Uint number.

when game wants to use the connect to GSTATS server, server will send an message to game which contains the challenge, the total length of message must bigger than 38bytes, and the challenge must bigger than 20bytes. when game received the challenge it will compute a response, the response is formed as follows. response = CRC32(<server challenge>,<length of server challenge>)||<game secret key> then game will compute the MD5 hash as MD5value = MD5(<response>,<length of response>) then encoded with Enctype3 then construct the challenge-response message as $\backslash auth\backslash\backslash gamename\backslash < gamename > \backslash response\backslash < MD5value > \backslash port\backslash < port > \backslash id\backslash < id >$

session key length (unknown) connction id = transfer ascii of sessionkey to integer

the initialization phase is finished. server challenge message length (bigger than 38-byte) server challenge length (bigger than 20-byte) $\backslash final\backslash$ is not encrypted using XOR Enctype1 at the end of the challenge that sends by the server.

20.2 Authenticate Player

20.2.1 Authenticate Player With Partner Information

Client request:

Code 20.2.1

\authp\\authtoken\<authtoken string>
\resp\response\<response string>\lid\<local id>\final\

Server response:

Server response is the same as 20.2.5.

20.2.2 Authenticate Player With Presence Connection Manager

Client request:

Code 20.2.2

\authp\\pid\<profile id>\resp\<response string>\lid\<local id>\final\

Server response:

Server response is the same as 20.2.5.

20.2.3 Authenticate Player With CD Key Hash

Client request:

Code 20.2.3

```
\authp\\nick\<nick name>\keyhash\<cd key hash>\resp\<response string>\lid\<local id>\final\
```

Server response:

Server response is the same as 20.2.5.

The challenge response string here is calculated from password and the connection id. xor challenge base string will be 0x38F371E6(decimal: 955478502)

Code 20.2.4

```
int temp = connid xor 0x38F371E6
string challenge;
string result;
for( int i=0; i<challenge.Lenth; i++ )
{
result+=( i + 17 + challenge[i]);
}
```

the calculation for resopnse string is connid xor 0x38F371E6

Game name	Secret key
Crysis2	8TTq4M

Table 20.1: Player authenticate response string

Server response:

Code 20.2.5

```
\pauthr\<profile id>\lid\<local id>\final\
```

20.3 Get Profileid

Client can get profile id by searching his cd-key hash in GSTATS server.

Client request:

Code 20.3.1

```
\getpid\\nick\<nick name>\keyhash\<cd key hash>\lid\<local id>\final\
```

Enum name	Description
pd_private_ro	Readable only by the authenticated client it belongs to, can only by set on the server.
pd_private_rw	Readable only by the authenticated client it belongs to, set by the authenticated client it belongs to.
pd_public_ro	Readable by any client, can only be set on the server.
pd_public_rw	Readable by any client, set by the authenicated client is belongs to.

Table 20.2: Persist storage enumerator

Server response:

Code 20.3.2

```
\getpidr\<profile id>\lid\<local id>\final\
```

20.4 Get Player Data

Client request:

Code 20.4.1

```
\getpd\\pid\<profile id>\ptype\<persist storage type>\dindex\<data index>\keys\final\
```

Server response:

Code 20.4.2

```
\getpdr\<profile id>\lid\<local id>\mod\<?>\length\<? length>\data\<player data>\final\
```

20.5 New Game

Creates a new game for logging and registers it with the stats server. Creates all the game structures, including buckets if needed.

Client request:

Code 20.5.1

```
\newgame\\connid\<connection id>\sesskey\<session key>\final\
```

Server response:

Code 20.5.2

20.6 Set Persist Data Helper

Client request:

Code 20.6.1

\setpd\\pid\<profile id>\ptype\<persist storage type> \dindex\<data index>\kv\<key value flag>\lid\<local id>\length\<size of game defined data>\final\

Server response:

Code 20.6.2

\setpdr\<success or fail>\lid\<local id>\pid\<profile id>
\mod\<modified time>\final\

20.7 Update Game Snapshot

Client request:

Old version:

Code 20.7.1

\updgame\\sesskey\<session key>\done\<final flag>
\gamedata\<game data>\final\

New version:

Code 20.7.2

\updgame\\sesskey\<session key>\connid\<connection id>
\done\<final flag>\gamedata\<game data>\final\

Server response:

Server only records the data.

Chapter 21

GameSpy Persist Storage

21.1 Introduction

If you store your data in keyelimited pairs, GetPersistDataValues will allow you to easily retrieve a subset of the stored data. To retrieve the entire data set, use GetPersistData. The data will be returned as a null-terminated string, unless no data is available (in which case len will be 0 in the callback).

21.2 Parameter

- localid: Your game-specific reference number for this player, returned in the callback to allow you to identify which player it is referring to.
- profileid: The profileid of the player whose data you are looking up. Returned by gpIDFromProfile() in the Presence & Messaging SDK, or using GetProfileIDFromCD
- type: The type of persistent data you are looking up
- index: Each profile can have multiple persistent data records associated with them. Usually you just want to use index 0.
- modifiedsince: A time value to limit the request for data. Data will only be returned if it has been modified since the time provided. If data has not been modified since that time, the callback will be called with a success value that indicates it is unmodified. Note: modification time is tracked for the given profileid/index, not on a per-persisttype or per-key basis keys: A "\" delimited list of the keys you want returned (for example: "\clan\color\homepage\birthday") PersDataCallbackFn: Callback that will be called with the data when it is returned instance: Pointer that will be passed to the callback function (for your use)

Part XV

GameSpy Chat Server

Chapter 22

Introduction

GameSpy Chat server is similar to IRC chat server, but support many extra commands. We use bracket <> to represent the data, in real protocol there is no bracket around each character field.
For example the following codes shows a welcome message.

Code 22.0.1

```
:www.retrospy.cc 001 spyguy :"Welcome to RetroSpy".
```

The client request formatted as follows:

Code 22.0.2

```
:<prefix> <middle> <command> <command params> :<tailing>
\r\n
```

22.1 Prefix Format

:<prefix> contains information as follows, when user is logged in the prefix will have user's information and server address. Before user logged in there will be only server address. Prefix using to indicate the message source.

Code 22.1.1

```
:<nick name>!<user name>@<server address>
or
:<server address>
```

22.2 Middle Format

In original IRC protocol, <middle> is used to represent the <command> and <command parameters>, however in GameSpy chat protocol they comprehend it a parameter before <command>, which cause the handler of sdk some time

skip the first parameter. The partial source code of **ciRplGetCKeyHandler()** we list as follows.

Code 22.2.1

```
channel = message->params[1];
nick = message->params[2];
cookie = message->params[3];
flags = message->params[4];
```

22.3 Command Format

GameSpy chat protocol contains two types of command, request is character command, response has both types of command.

- Enumerate command
- Character command

22.3.1 Enumerate Command

The format of enumerate commands are present as three digit numbers **XXX**. The commands which used by GameSpy chat server are list as Table 22.2.

Name	Value	Description
RPL_WELCOME	”001”	Welcome reply
RPL_USRIP	”302”	Get user’s public ip reply
RPL_WHOISUSER	”311”	Search for user information reply
RPL_ENDOFWHOIS	”318”	End of search user information reply
RPL_WHOISCHANNELS	”319”	Search channel information reply
RPL_LISTSTART	”321”	Start list channels reply
RPL_LIST	”322”	List channel information reply
RPL_LISTEND	”323”	End of list channel information reply
RPL_CHANNELMODEIS	”324”	Get channel modes reply
RPL_NOTOPIC	”331”	There is no topic for this channel reply
RPL_TOPIC	”332”	Get channel topic reply
RPL_WHOREPLY	”352”	Search users reply
RPL_ENDOFWHO	”315”	The end of Search users reply
RPL_NAMEREPLY	”353”	Get users in channel reply
RPL_ENDOFNAMES	”366”	End of list users in this channel reply
RPL_BANLIST	”367”	Get ban list reply
RPL_ENDOFBANLIST	”368”	End of ban list reply
RPL_GETKEY	”700”	Get user value by specific keys reply
RPL_ENDGETKEY	”701”	End of GETKEY reply
RPL_GETCKEY	”702”	Get channel or user’s values reply
RPL_ENDGETCKEY	”703”	End of GETCKEY reply
RPL_GETCHANKEY	”704”	Get channel values reply
RPL_SECUREKEY	”705”	Use gamespy encryption to communicate
RPL_CDKEY	”706”	Authenticate with cdkey reply
RPL_LOGIN	”707”	Login as a GameSpy user reply
RPL_GETUDPRELAY	”712”	

Table 22.2: The enumerate reply command

The 3 digits number will convert to ASCII code and send to client.

22.3.2 Character Command

The other command is character command, the format of these commands are using words to represent command.

Name	Value	Description

Name	Value	Description
Private message response	”PRIVMSG”	
Notice message response	”NOTICE”	
Under the table message response	”UTM”	
Above the table message response	”ATM”	
Ping message response	”PING”	
Nick message response	”NICK”	
Join channel response	”JOIN”	
Leave channel response	”PART”	
Kick player response	”KICK”	
Quit response	”QUIT”	
Kill response	”KILL”	
Get or set topic response	”TOPIC”	
Get or set mode response	”MODE”	
Error message	”ERROR”	
Invite Response	”INVITE”	

Table 22.3: Character reply commands

22.4 Command Parameters

Command parameters are the extra information that each request contains. Such as nick name, user name, channel name etc.

22.5 Tailing Format

The start of <tailing> should be :, the space inside <tailing> is ignored by server and client, which means you can send <:hello my friend> to others. usually <tailing> is used to carry some messages.

Appendix A

Login Proof Challenge Generation Algorithm

Appendix B

Gstats Initial Encryption

Appendix C

CDKey Server Initial Encryption

Appendix D

GameSpy Secret Key

If a game is using GameSpy service, GameSpy will issue a secret key to the game, which length is at 5-byte. The secret key that GameSpy issued is not the traditional secret key in public-key cryptography, actually it is an key for simple symmetric encryption.

Game name	Secret key
Crysis2	8TTq4M

Table D.1: Secret key example