Spring lobby protocol description

Introduction

NOTE: This document is available in a HTML form at the following address: *http://springrts.com/lobby/protocol/ProtocolDescription.xml*

This is a simple text protocol in a "human readable form". I decided for text protocol since it's easier to debug and implement on the client side. It does require some more overhead, but since server and clients do not communicate much anyway, this is not an issue. What really eats the bandwidth is the game itself, not lobby.

Single word arguments are separated by spaces, "sentences" (a sentence is a series of words separated by spaces, denoting a single argument) are separated by TABs. Don't forget to replace any TABs in your sentences with spaces! If you don't, your command is invalid and will not be read correctly by server.

A client must maintain a constant connection with server, that is if no data is to be transfered, client must send a PING command to the server on regular intervals. This ensures server to properly detect any network timeouts/disconnects. Server will automatically disconnect a client if no data has been received from it within a certain time period. Client should send some data at least on half of that interval.

Whether battle is in progress or not, clients may find out by monitoring battle's founder status. If his status changes to "in game", battle's status is also "in game".

Client's status and battle status are assumed to be 0 if not said otherwise. Each client should assume that every other client's status and battle status are 0 if not stated otherwise by the server. So when client joins the server, he is notified about statuses of only those users, who have status or battle status different from 0. This way we also save some bandwidth (although that is not really an issue here. If we were worried about this protocol taking up too much bandwidth, we wouldn't use text protocol, now would we?).

I use two terms for "clients": a client and an user. At first user was considered to be a registered client, that is client who is logged-in. I started to mix the two terms after a while but that should not confuse you.

Most of the commands that notify about some changes are forwarded to the source of the change too. For example: When client sends MYBATTLESTATUS command, server sends CLIENTBATTLESTATUS command to all users in the battle, including the client who sent MYBATTLESTATUS command. This is a good practice, because this way client can synchronize his local status with server's after server has actually updated his status (serves also as confirmation).

All commands except for some admin specific commands are listed here. See "AdminCommands.txt" for those. For more details on specific command, see tryToExecCommand() method in TASServer.java.

Also, check notes (under "--- NOTES ---" field) in TASServer.java since they contain some information and tricks on implementing this protocol.

As of version 0.33, command syntax has been slightly changed. Messages may now contain IDs: Any message may be prefixed with a message ID. For an example: "#123 SAY main hello!"
Server will use this ID with a reply, for an example: "#123 SAID main Betalord hello!"
The message ID has only one function: to ease tracking server responses for the client. Using message IDs means that client can also use synchronous communication (with certain commands), which makes it easier to implement certain commands that require multiple responses in both directions.
Note that IDs should be non-negative values, in 31-bit range (0..2147483647). Negative values are reserved for internal handling logic of server / lobby clients.

Legend:

- {} = sentence = several words (at least one!) separated by space characters
- {} {} ... = two or more sentences. They are separated by TAB characters.
- words are always separated by space characters.
- | = separates choices ("OR")
- [] = denotes an optional argument

Some statistics on this document:

- Number of commands described: 118
- Number of client commands: 49
- Number of server commands: 69

Recent changes

Version 0.36:

- scriptPassword argument added: This change affects the following commands: LOGIN, JOINBATTLE, JOINEDBATTLE.

Version 0.35:

- New 'servermode' argument has been added to TASSERVER command.
- Added SETSCRIPTTAGS command. This command replaces UPDATEBATTLEDETAILS command.
- Added REMOVESCRIPTTAGS command, companion to SETSCRIPTTAGS.
- New startpos type added: "choose before battle" (type 3). These change affects following commands: OPENBATTLE, JOINBATTLE.
- Added two more in-game ranks, one at 300h, the other at 1000h.
- Removed UPDATEBATTLEDETAILS command as it is no longer needed (use SETSCRIPTTAGS instead). Also removed several arguments from JOINBATTLE and OPENBATTLE commands (all these arguments are handled via SETSCRIPTTAGS

command now). See descriptions of these commands for more info.

Version 0.34:

- Message/command ID system has been implemented to automatically (and transparently) support any command regardless of the method handler implementation. This means that the response to any command will always contain message ID if you specifed it with the original command.
- PING command no longer supports 'key' argument since it is not needed anymore (use message ID instead)
- Lobby programs should now acquire Spring version via unitsync.dll and not by running spring.exe, although both ways are legal.
- LOGIN command now accepts additional parameter 'userID', which is optional. See LOGIN command description for more info.
- Added ACQUIREUSERID and USERID commands. See documentation bellow for further details.
- Added 'url' parameter to SERVERMSGBOX command.
- Added lineage mode to "end game condition" parameter (which is used with several commands).

Version 0.33:

- TASSERVER command has been modified, see documentation bellow for further details. Note that clients should now acquire local Spring version by running "spring.exe /version" (they need this info to compare local Spring version to the server's).
- REQUESTUPDATEFILE command has been modified, see documentation bellow for further details
- Note that MsgIDs (as described in the "Introduction" part) aren't fully supported yet - server will parse IDs, but won't include them in the response.

Command list

PING *Source: client*

Description

Client should send this command on every few seconds to maintain constant connection to the server. Server will assume timeout occured if it does not hear from client for more than x seconds. To figure out how long does a reply take, use message ID with this command.

Response

See PONG.

PONG *Source: server*

Description

Used as a response to a PING command. You can determine how long did the reply took by using a message ID with the PING command.

TASSERVER server_version spring_version udpport servermode *Source: server*

Description

This is the first message (i.e. "greeting message") that client receives upon connecting to the server.

server_version: This is server's version number (client should check if it supports this server version before attempting to log in. If it does not support it, it should send REQUESTUPDATEFILE command).

spring_version: This is the latest Spring version which user needs in order to play with other players on this server. Client should check if it has this Spring version before attempting to log in. If it does not have it, it should send a REQUESTUPDATEFILE command.
Note that if the value of this parameter is "*", client should simply ignore it since this means that server does not contain any updates nor does it require latest Spring version (this is usually so when server is running in LAN mode).

udpport: This is server's UDP port where "NAT Help Server" is running. This is the port to which clients should send their UDP packets when trying to figure out their public UDP source port. This is used with some NAT traversal techniques (e.g. "hole punching").

servermode: Tells what mode server is in. Currently valid are:

- 0 - normal mode
- 1 - LAN mode

Examples

TASSERVER 0.35 0.72b1 8201 0
TASSERVER 0.35 * 8201 0

REQUESTUPDATEFILE {name and version} *Source: client*

Description

Sent by client who is using an outdated Spring or lobby program and is trying to figure out if server has some update for him (client can figure out if he is using outdated software via TASSERVER command). If both, Spring and lobby program are outdated, request only Spring update since it will most likely contain an update for the lobby

program as well.

name and version: Program name and version for which client is requesting an update. For example, if user's Spring is outdated it will provide for example "Spring 0.72b1", or if TASClient is outdated it will provide for example "TASClient 0.30", similar for other lobby clients or bots, for which server provides updates.

Response

Server will respond with OFFERFILE command if it has an update for the client, or else it will display a message box (via SERVERMSGBOX command) or similar.

Examples

REQUESTUPDATEFILE Spring 0.72b1
REQUESTUPDATEFILE TASClient 0.30

REGISTER username password *Source: client*

Description

Client sends this command when trying to register a new account. Note that client mustn't already be logged in, or else server will deny his request. If server is running in LAN_MODE, this command will be ignored.

password: Must be sent in encoded form (MD5 hash in base-64 form).

Response

Server will respond with either REGISTRATIONDENIED or REGISTRATIONACCEPTED command.

Examples

REGISTER Johnny Gnmk1g3mcY6OWzJuM4rlMw==

REGISTRATIONDENIED {reason} *Source: server*

Description

Sent to client who has just sent REGISTER command, if registration has been refused.

REGISTRATIONACCEPTED *Source: server*

Description

Sent to client who has just sent REGISTER command, if registration has been accepted.

RENAMEACCOUNT newUsername *Source: client*

Description

Will rename current account which is being used by the user to newUsername. User has to be logged in for this to work. After server renames the account, it will disconnect him.

Response

No response is prescribed, although server may reply with some SERVERMSG command.

Examples

RENAMEACCOUNT Johnny2

CHANGEPASSWORD oldPassword newPassword *Source: client*

Description

Will change password of client's account (which he is currently using).

Response

No response is prescribed, although server may reply with some SERVERMSG command.

LOGIN username password cpu localIP {lobby name and version} [{userID}] [{compFlags}] *Source: client*

Description

Sent by client when he is trying to log on the server. Server may respond with ACCEPTED or DENIED command. Note that if client hasn't yet confirmed the server agreement, then server will send the agreement to client upon receiving LOGIN command (LOGIN command will be ignored - client should resend LOGIN command once user has agreed to the agreement or disconnect from the server if user has rejected the agreement).
Also see LOGININFOEND command.

password: Should be sent in encoded form (MD5 hash in base-64 form). Note that when server is running in lan mode, you can specify any username and password (password will be ignored, but you must send some string anyway - you mustn't ommit it!)

cpu: An integer denoting the speed of client's processor in MHz (or value of x+ tag if AMD). Client should leave this value at 0 if it can't figure out its CPU speed.

localIP: As localIP client should send his local IP (e.g. 192.168.x.y, or whatever it uses) so server can forward local IPs to clients behind same NAT (this resolves some of the host/joining issues). If client is unable to determine his local IP, he should send "*" instead.

userID: This is a unique user identification number provided by the client-side software. It should be an unsigned integer encoded in hexadecimal form (see examples). Note that this parameter is optional - by default it is not used/set. Server will send a ACQUIREUSERID command to tell the client that he must provide a user ID, if needed. However, if client-side lobby program was using user ID before, it should send it along with LOGIN command.

compFlags: When connecting, the lobby client can tell the lobby server it is compatible with some optional functionalities that break backward compatibility. Each flag in the space separated compFlags parameter indicates a specific functionality (like IRC user/channel flags). By default, all the optional functionalities are considered as not supported by the client. The currently compatibility flags are:
'a': client supports accountIDs sent in ADDUSER command
'b': client supports battle authorization system (JOINBATTLEREQUEST, JOINBATTLEACCEPT and JOINBATTLEDENY commands)
'sp': client supports scriptPassword sent in the JOINEDBATTLE command

Examples

```
LOGIN Johnny Gnmk1g3mcY6OWzJuM4rlMw== 3200 192.168.1.100 TASClient 0.30
LOGIN Johnny Gnmk1g3mcY6OWzJuM4rlMw== 3200 * TASClient 0.30
LOGIN Johnny Gnmk1g3mcY6OWzJuM4rlMw== 3200 * TASClient 0.30 FA23BB4A
LOGIN Johnny Gnmk1g3mcY6OWzJuM4rlMw== 3200 * TASClient 0.30 0 ab
```

ACCEPTED username *Source: server*

Description

Sent as a response to LOGIN command if it succeeded. After server has finished sending info on clients and battles, it will send LOGININFOEND command indicating that it has finished sending the login info.

DENIED {reason} *Source: server*

Description

Sent as a response to a failed LOGIN command.

LOGININFOEND *Source: server*

Description

Sent by server indicating that it has finished sending the login info (which is sent immediately after accepting the LOGIN command). This way client can figure out when server has finished updating clients and battles info and can so figure out when login sequence is finished.

Note that sending login info consists of 4 phases:

- sending MOTD
- sending list of all users currently logged on the server
- sending info currently active battles
- sending statuses of all users

LOGININFOEND command is sent when server has finished sending this info.

AGREEMENT {agreement} *Source: server*

Description

Sent by server upon receiving LOGIN command, if client has not yet agreed to server's "terms-of-use" agreement. Server may send multiple AGREEMENT commands (which corresponds to multiple new lines in agreement), finishing it by AGREEMENTEND command. Client should send CONFIRMAGREEMENT and then resend LOGIN command, or disconnect from the server if he has chosen to refuse the agreement.

agreement: Agreement is sent in "Rich Text" format (.rtf file streamed via socket).

Response

No response is expected until AGREEMENTEND is received. See that command for more info.

AGREEMENTEND *Source: server*

Description

Sent by server after multiple AGREEMENT commands. This way server tells the client that he has finished sending the agreement (this is the time when lobby should popup the "agreement" screen and wait for user to accept/reject it).

Response

CONFIRMAGREEMENT command is expected from the client in case user agreed to the agreement. Or else client should disconnect from the server.

CONFIRMAGREEMENT *Source: client*

Description

Sent by client notifying the server that user has confirmed the agreement. Also see AGREEMENT command.

MOTD {message} *Source: server*

Description

Sent by server after client has successfully logged in. Server can send multiple MOTD commands (each MOTD corresponds to one line, for example).

OFFERFILE options {filename} {url} {description} *Source: server*

Description

Sent as a response to REQUESTUPDATEFILE command.

options: An integer (in text format), where each bit has its own meaning:

- bit0 - should file be opened once it is downloaded
- bit1 - should program be closed once file is downloaded
- bit2 - is this file "mandatory" (e.g. patch - if user doesn't accept it, it must disconnect from the server)

filename: Filename is the name of the file to which the file should be saved on client's side. If filename = *, then client should extract file name from URL and use it instead of filename.

url: URL from which lobby program should download the update file.

description: Description of the file offered. Lobby program should display this description (in a message box, for example).

Examples

OFFERFILE 7 * http://taspring.clan-sy.com/dl/spring_0.74b2_update.exe This is a 0.74b1->0.74b2 patch.

UDPSOURCEPORT port *Source: server*

Description

Sent as a response to client's UDP packet (used with "hole punching" NAT traversal technique). For more info see the description of NAT traversal technique used by the lobby.

Examples

UDPSOURCEPORT 52361

CLIENTIPPORT username ip port *Source: server*

Description

Sent to battle's host notifying him about another client's IP address and his public UDP source port. Host needs to know public UDP source ports of all players in his battle in order for "hole punching" technique to work. IP is needed with both nat traversal methods ("hole punching" and "fixed source ports"). This command is sent to the battle host immediately after new user joins the battle (currently only if some nat traversal technique is used).

HOSTPORT port *Source: server*

Description

Sent by server to all clients participating in the battle (except for the host) notifying them about the new host port. This message will only be sent right before host starts a game, if this battle is being hosted using "hole punching" NAT traversal technique. When client receives this message, he should replace battle's host port with this new port, so that when game actually starts, he will connect to this new port rather than the port which host selected when he initialy started the battle (with OPENBATTLE command).

SERVERMSG {message} *Source: server*

Description

A general purpose message sent by the server. Lobby program should display it either in the chat log or in some kind of system log, where client will notice it. Also see SERVERMSGBOX command.

Examples

SERVERMSG Server is going down in 5 minutes for a restart, due to a new update.

SERVERMSGBOX {message} [{url}] *Source: server*

Description

This is a message sent by the server that should open in a message box dialog window (and not just in the console). Also see SERVERMSG command.

url: If specified, lobby client should ask user if he wants this url to be opened in a browser window. Lobby client should then (once user clicks OK) launch the default browser and open this url in it.

ADDUSER username country cpu [accountID] *Source: server*

Description

Sent by server to client telling him new user joined a server. Client should add this user to his clients list which he must maintain while he is connected to the server. Server will send multiple commands of this kind once client logs in, sending him the list of all users currently connected.

country: A two-character country code based on ISO 3166 standard. See http://www.iso.org/iso/en/prods-services/iso3166ma/index.html

cpu: See LOGIN command for CPU field.

accountID: Unique ID bound to the account (can be used to track renammed accounts). This field is only provided if the lobby client sent the accountID compatibility flag ('a') in the LOGIN command.

REMOVEUSER username *Source: server*

Description

Sent to client telling him a user disconnected from server. Client should remove this user from his clients list which he must maintain while he is connected to the server.

JOIN channame [key] *Source: client*

Description

Sent by client trying to join a channel.

key: If channel is locked, then client must supply a correct key to join the channel (clients with access >= Account.ADMIN_ACCESS can join locked channels withouth supplying the key - needed for ChanServ bot).

Examples

JOIN main
JOIN myprivatechannel mypassword

JOIN channame *Source: server*

Description

Sent to a client who has successfully joined a channel.

channame: Name of the channel to which client has just joined (by previously sending the JOIN command).

Examples

JOIN main

JOINFAILED channame {reason} *Source: server*

Description

Sent if joining a channel failed for some reason.

reason: Always provided.

CHANNELS *Source: client*

Description

Sent by client when requesting channels list

Response

Server will respond with a series of CHANNEL command, ending it with ENDOFCHANNELS command.

Examples

JOIN main

CHANNEL channame usercount *Source: server*

Description

Sent by server to client who requested channel list via CHANNELS command. A series of these commands will be sent to user, one for each open channel. Topic parameter may be omited if topic is not set for the channel. A series of CHANNEL commands is ended by ENDOFCHANNELS command.

usercount: Number of users in channel.

ENDOFCHANNELS *Source: server*

Description

Sent to client who previously requested channel list, after a series of CHANNEL commands (one for each channel).

MUTELIST channame *Source: client*

Description

Sent by client when requesting mute list of a channel.

Examples

MUTELIST main

MUTELISTBEGIN channame *Source: server*

Description

Sent by server when sending channel's mute list to the client. This command is immediately followed by 0 or more MUTELIST commands and finally by a MUTELISTEND command.

Response

Followed by 0 or more MUTELIST commands and finally by a MUTELISTEND command.

MUTELIST {mute description} *Source: server*

Description

Sent after MUTELISTBEGIN command. Multiple commands of this kind may be sent after MUTELISTSTART command (or none, if mute list is empty).

mute description: Form of this argument is not prescribed (it may vary from version to version). Lobby program should simply display it as it receives it.

Examples

MUTELIST Johnny, 345 seconds remaining
MUTELIST rabbit, indefinite time remaining

MUTELISTEND *Source: server*

Description

Sent by server after it has finished sending the list of mutes for a channel. Also see MUTELIST and MUTELISTBEGIN commands.

CHANNELTOPIC channame author changedtime {topic} *Source: server*

Description

Sent to client who just joined the channel, if some topic is set for this channel.

changedtime: Time in milliseconds since Jan 1, 1970 (UTC). Divide by 1000 to get unix time.

CHANNELTOPIC channame {topic} *Source: client*

Description

Sent by privileged user who is trying to change channel's topic. Use * as topic if you wish to disable it.

CLIENTS channame {clients} *Source: server*

Description

Sent to a client who just joined the channel. Note: Multiple commands of this kind can be sent in a row. Server takes the list of clients in a channel and separates it into several lines and sends each line seperately. This ensures no line is too long, because client may have some limit set on the maximum length of the line and it could ignore it if it was too long. Also note that the client itself (his username) is sent in this list too! So when client receives JOIN command he should not add his name in the clients list of the channel - he should wait for CLIENTS command which will contain his name too and he will add himself then automatically.

JOINED channame username *Source: server*

Description

Sent to all clients in a channel (except the new client) when a new client joins the channel.

LEAVE channame *Source: client*

Description

Sent by client when he is trying to leave a channel. When client is disconnected, he is automatically removed from all channels.

LEFT channame username [{reason}] *Source: server*

Description

Sent by server to all clients in a channel when some client left this channel. WARNING: Server does not send this command to a client that has just left a channel, because there is no need to (client who has left the channel knows that already). Client that was kicked from the channel is notified about it via FORCELEAVECHANNEL command.

FORCELEAVECHANNEL channame username [{reason}] *Source: client*

Description

Sent by client (moderator) requsting that the user is removed from the channel. User will be notified with FORCELEAVECHANNEL command.

FORCELEAVECHANNEL channame username [{reason}] *Source: server*

Description

Sent to user who has just been kicked from the channel #channame by user "username". (lobby client should now internally close/detach from the channel as he was removed from the clients list of #channame on server side)

CHANNELMESSAGE channame message *Source: server*

Description

Sent by server to all clients in a channel. Used to broadcast messages in the channel.

SAY channame {message} *Source: client*

Description

Sent by client when he is trying to say something in a specific channel. Client must first join the channel before he can receive or send messages to that channel.

Response

See SAID command.

SAID channame userame {message} *Source: server*

Description

Sent by server to all clients participating in this channel when one of the clients sent a message to it (including the author of this message).

SAYEX channame {message} *Source: client*

Description

Sent by any client when he is trying to say something in "/me" irc style. Also see SAY command.

SAIDEX channame userame {message} *Source: server*

Description

Sent by server when client said something using SAYEX command.

SAYPRIVATE username {message} *Source: client*

Description

Sent by client when he is trying to send a private message to some other client.

Response

Server will respond with a SAYPRIVATE command.

SAYPRIVATE username {message} *Source: server*

Description

Sent by server to a client who just sent SAYPRIVATE command to server. This way client can verify that server did get his message and that receiver will get it too.

SAIDPRIVATE username {message} *Source: server*

Description

Sent by server when some client sent this client a private message.

OPENBATTLE type natType password port maxplayers hashcode rank maphash {map} {title} {modname} *Source: client*

Description

Sent by client when he is trying to open a new battle. The client becomes a founder of this battle, if command is successful (see Response section).

type: Can be 0 or 1 (0 = normal battle, 1 = battle replay)

natType: NAT traversal method used by the host. Must be one of: 0: none 1: Hole punching 2: Fixed source ports

password: Must be "*" if founder does not wish to have password-protected game.

hashcode: A signed 32-bit integer (acquired via unitsync.dll).

maphash: A signed 32-bit integer as returned from unitsync.dll.

Response

Client is notified about this command's success via OPENBATTLE/OPENBATTLEFAILED commands.

OPENBATTLE BATTLE_ID *Source: server*

Description

Sent to a client who previously sent OPENBATTLE command to server, if client's request to open new battle has been approved. If server rejected client's request, client is notified via OPENBATTLEFAILED command. Server first sends BATTLEOPENED command, then OPENBATTLE command (this is important - client must have the battle in his battle list before he receives OPENBATTLE command!).

BATTLEOPENED BATTLE_ID type natType founder IP port maxplayers passworded rank maphash {map} {title} {modname} *Source: server*

Description

Sent by server to all registered users, when a new battle has been opened. Series of BATTLEOPENED commands are sent to user when he logs in (1 command for each battle). Use Battle.createBattleOpenedCommand method to create this command in a String (when modifying TASServer source).

type: Can be 0 or 1 (0 = normal battle, 1 = battle replay)

natType: NAT traversal method used by the host. Must be one of: 0: none 1: Hole punching 2: Fixed source ports

founder: Username of the client who started this battle.

passworded: A boolean - must be "0" or "1" and not "true" or "false" as it is default in Java! Use Misc.strToBool and Misc.boolToStr methods (from TASServer source) to convert from one to another.

maphash: A signed 32-bit integer as returned from unitsync.dll.

BATTLECLOSED BATTLE_ID *Source: server*

Description

Sent when founder has closed a battle (or if he was disconnected).

JOINBATTLE BATTLE_ID [password] [scriptPassword] *Source: client*

Description

Sent by a client trying to join a battle. Password is an optional parameter.

scriptPassword: A random, client-generated string wich will be written to script.txt by the host, to avoid account spoofing (= someone is trying to join the battle under wrong user-name). If set, the password has to be set too, even if it is empty.

JOINBATTLE BATTLE_ID hashcode *Source: server*

Description

Sent by server telling the client that he has just joined the battle successfully. Server will also send a series of CLIENTBATTLESTATUS commands after this command, so that user will get the battle statuses of all the clients in the battle.

hashcode: A signed 32-bit integer.

JOINBATTLEREQUEST username ip *Source: server*

Description

Sent by server to battle founder each time a client requests to join his battle, if the battle founder supports battle authorization system (i.e. he provided the battle authorization compatibility flag ('b') in the LOGIN command.).

Response

When client receives this command, he must send either a JOINBATTLEACCEPT or a JOINBATTLEDENY command to the server.

JOINBATTLEACCEPT username *Source: client*

Description

Sent by client in response to a JOINBATTLEREQUEST command in order to allow the user to join the battle.

JOINBATTLEDENY username [{reason}] *Source: client*

Description

Sent by client in response to a JOINBATTLEREQUEST command in order to prevent the user from joining the battle.

JOINEDBATTLE BATTLE_ID username [scriptPassword] *Source: server*

Description

Sent by server to all clients when a new client joins the battle.

scriptPassword: A random, client-generated string wich will be written to script.txt by the host, to avoid account spoofing (= someone is trying to join the battle under wrong user-name).

LEFTBATTLE BATTLE_ID username *Source: server*

Description

Sent by server to all users when client left a battle (or got disconnected from the server).

LEAVEBATTLE *Source: client*

Description

Sent by the client when he leaves a battle. Also sent by a founder of the battle when he closes the battle.

JOINBATTLEFAILED {reason} *Source: server*

Description

Sent by server to user who just tried to join a battle but has been rejected by server.

OPENBATTLEFAILED {reason} *Source: server*

Description

Sent by server to user who just tried to open(=host) a new battle and was rejected by the server.

UPDATEBATTLEINFO BATTLE_ID SpectatorCount locked maphash {mapname} *Source: server*

Description

Sent by server to all registered clients telling them some of the parameters of the battle changed. Battle's inside changes, like starting metal, energy, starting position etc., are sent only to clients participating in the battle via SETSCRIPTTAGS command.

SpectatorCount: Assume that spectator count is 0 if battle type is 0 (normal battle) and 1 if battle type is 1 (battle replay), as founder of the battle is automatically set as a spectator in that case.

locked: A boolean (0 or 1). Note that when client creates a battle, server assumes it is unlocked (by default). Client must make sure it actually is.

maphash: A signed 32-bit integer. See OPENBATTLE command for more info.

UPDATEBATTLEINFO SpectatorCount locked maphash {mapname} *Source: client*

Description

Sent by the founder of the battle telling the server some of the "outside" parameters of the battle changed.

locked: A boolean (0 or 1). Note that when client creates a battle, server assumes it is unlocked (by default). Client must make sure it actually is.

maphash: A signed 32-bit integer. See OPENBATTLE command for more info.

mapname: Must NOT contain file extension!

SAYBATTLE {message} *Source: client*

Description

Sent by client who is participating in a battle to server, who forwards this message to all other clients in the battle. BATTLE_ID is not required since every user can participate in only one battle at the time. If user is not participating in the battle, this command is ignored and is considered invalid.

SAIDBATTLE username {message} *Source: server*

Description

Sent by server to all clients participating in a battle when client sent a message to it using SAYBATTLE command. BATTLE_ID is not required since every client knows in which battle he is participating in, since every client may participate in only one battle at the time. If client is not participating in a battle, he should ignore this command or raise an error (this should never happen!).

SAYBATTLEEX {message} *Source: client*

Description

Sent by any client participating in a battle when he wants to say something in "/me" irc style. Server can forge this command too (for example when founder of the battle kicks a user, server uses SAYBATTLEEX saying founder kicked a user).

SAIDBATTLEEX username {message} *Source: server*

Description

Sent by server to all clients participating in a battle when client used SAYBATTLEEX command. See SAYBATTLEEX for more info.

MYSTATUS status *Source: client*

Description

Sent by client to server telling him his status changed. To figure out if battle is "in-game", client must check in-game status of the host.

status: A signed integer in text form (e.g. "1234"). Each bit has its meaning:

- b0 = in game (0 - normal, 1 - in game)
- b1 = away status (0 - normal, 1 - away)
- b2-b4 = rank (see Account class implementation for description of rank) - client is not allowed to change rank bits himself (only server may set them).
- b5 = access status (tells us whether this client is a server moderator or not) - client is not allowed to change this bit himself (only server may set them).
- b6 = bot mode (0 - normal user, 1 - automated bot). This bit is copied from user's account and can not be changed by the client himself. Bots differ from human players in that they are fully automated and that some anti-flood limitations do not apply to them.

CLIENTSTATUS username status *Source: server*

Description

Sent by server to all registered clients indicating that client's status changed. Note that client's status is considered 0 if not said otherwise (for example, when you logon, server sends only statuses of those clients whose statuses differ from 0, to save the bandwidth).

status: See MYSTATUS command for possible values of this parameter.

MYBATTLESTATUS battlestatus myteamcolor *Source: client*

Description

Sent by a client to the server telling him his status in the battle changed.

battlestatus: An integer but with limited range: 0..2147483647 (use signed int and consider only positive values and zero). Number is sent as text. Each bit has its meaning:

- b0 = undefined (reserved for future use)
- b1 = ready (0=not ready, 1=ready)
- b2..b5 = team no. (from 0 to 15. b2 is LSB, b5 is MSB)
- b6..b9 = ally team no. (from 0 to 15. b6 is LSB, b9 is MSB)
- b10 = mode (0 = spectator, 1 = normal player)
- b11..b17 = handicap (7-bit number. Must be in range 0..100). Note: Only host can change handicap values of the players in the battle (with HANDICAP command). These 7 bits are always ignored in this command. They can only be changed using HANDICAP command.
- b18..b21 = reserved for future use (with pre 0.71 versions these bits were used for team color index)
- b22..b23 = sync status (0 = unknown, 1 = synced, 2 = unsynced)
- b24..b27 = side (e.g.: arm, core, tll, ... Side index can be between 0 and 15, inclusive)
- b28..b31 = undefined (reserved for future use)

myteamcolor: Should be 32-bit signed integer in decimal form (e.g. 255 and not FF) where each color channel should occupy 1 byte (e.g. in hexdecimal: $00BBGGRR, B = blue, G = green, R = red). Example: 255 stands for $000000FF.

CLIENTBATTLESTATUS username battlestatus teamcolor *Source: server*

Description

Sent by server to users participating in a battle when one of the clients changes his battle status.

battlestatus: See MYBATTLESTATUS command for possible values of this parameter.

teamcolor: Uses same format as the one used with MYBATTLESTATUS command.

REQUESTBATTLESTATUS *Source: client*

Description

Sent by server to user who just opened a battle or joined one. This command is sent after all CLIENTBATTLESTATUS commands for all clients have been sent. This way user can choose suitable team, ally and color numbers since he knows battle statuses of other clients already.

Response

When client receives this command, he must send MYBATTLESTATUS command to the server so that server can synchronize battle status with user's.

HANDICAP username value *Source: client*

Description

Sent by founder of the battle changing username's handicap value (of his battle status). Only founder can change other users handicap values (even they themselves can't change it).

value: Must be in range [0, 100] (inclusive).

KICKFROMBATTLE username *Source: client*

Description

Sent by founder of the battle when he kicks the client out of the battle. Server remove client from the battle and notify him about it via FORCEQUITBATTLE command.

FORCEQUITBATTLE *Source: server*

Description

Sent to client for whom founder requested kick with KICKFROMBATTLE command. Client doesn't need to send LEAVEBATTLE command, that is already done by the server. The only purpose this commands serves to is to notify client that he was kicked from the battle. Note that client should close the battle internally, since he is no longer a part of it (or he can do that once he receives LEFTBATTLE command containing his username).

FORCETEAMNO username teamno *Source: client*

Description

Sent by founder the of battle when he is trying to force some other client's team number to 'teamno'. Server will update client's battle status automatically.

FORCEALLYNO username teamno *Source: client*

Description

Sent by founder of the battle when he is trying to force some other client's ally number to 'allyno'. Server will update client's battle status automatically.

FORCETEAMCOLOR username color *Source: client*

Description

Sent by founder of the battle when he is trying to force some other client's team color to 'color'. Server will update client's battle status automatically.

color: Should be a 32-bit signed integer in decimal form (e.g. 255 and not FF) where each color channel should occupy 1 byte (e.g. in hexdecimal: $00BBGGRR, B = blue, G = green, R = red). Example: 255 stands for $000000FF.

FORCESPECTATORMODE username *Source: client*

Description

Sent by founder of the battle when he is trying to force some other client's mode to spectator. Server will update client's battle status automatically.

DISABLEUNITS unitname1 unitname2 ... *Source: client*

Description

Sent by founder of the battle to server telling him he disabled one or more units. At least one unit name must be passed as an argument.

unitname1: Multiple units may follow, but at least one must be present in the arguments list.

DISABLEUNITS unitname1 unitname2 ... *Source: server*

Description

Sent by server to all clients in the battle except for the founder, notifying them some units have been added to disabled units list. Also see DISABLEUNITS

unitname1: Multiple units may follow, but at least one must be present in the arguments list.

ENABLEUNITS unitname1 unitname2 ... *Source: client*

Description

Sent by founder of the battle to server telling him he enabled one or more previous disabled units. At least one unit name must be passed as an argument.

unitname1: Multiple units may follow, but at least one must be present in the arguments list.

ENABLEUNITS unitname1 unitname2 ... *Source: server*

Description

Sent by server to all clients in the battle except for the founder, notifying them some units have been removed from disabled units list.

unitname1: Multiple units may follow, but at least one must be present in the arguments list.

ENABLEALLUNITS *Source: client*

Description

Sent by founder of the battle to server telling him he enabled ALL units and so clearing the disabled units list.

ENABLEALLUNITS *Source: server*

Description

Sent by server to all clients in the battle except for the founder, telling them that disabled units list has been cleared.

RING username *Source: client*

Description

Sent by client to server when trying to play a "ring" sound to user 'username'. Only privileged users can ring anyone, although "normal" clients can ring only when they are hosting and only players participating in their battle.

RING username *Source: server*

Description

Sent by server to client telling him user 'username' just rang (client should play the "ring" sound once he receives this command).

REDIRECT ipaddress *Source: server*

Description

Sent by server when in "redirection mode". When client connects, server will send him only this message and disconnect the socket immediately. Client should connect to 'ipaddress' in that case. This command may be useful when official server address changes, so that clients are automatically redirected to the new one.

Examples

REDIRECT 87.96.164.14

BROADCAST {message} *Source: server*

Description

Sent by server when urgent message has to be delivered to all users. Lobby program should display this message either in some system log or in some message box (user must see this message!).

ADDBOT name battlestatus teamcolor {AIDLL} *Source: client*

Description

With this command client can add bots to the battle.

teamcolor: Should be 32-bit signed integer in decimal system (e.g. 255 and not FF) where each color channel should occupy 1 byte (e.g. in hexdecimal: $00BBGGRR, B = blue, G = green, R = red).

ADDBOT BATTLE_ID name owner battlestatus teamcolor {AIDLL} *Source: server*

Description

This command indicates that client has added a bot to the battle.

BATTLE_ID: BATTLE_ID is there just to help client verify that the bot is meant for his battle.

teamcolor: Should be 32-bit signed integer in decimal system (e.g. 255 and not FF) where each color channel should occupy 1 byte (e.g. in hexdecimal: $00BBGGRR, B = blue, G = green, R = red).

REMOVEBOT name *Source: client*

Description

Removes a bot from the battle.

REMOVEBOT BATTLE_ID name *Source: server*

Description

Indicates that bot has been removed from the battle. Sent by server.

BATTLE_ID: BATTLE_ID is there just to help client verify that the bot is meant for his battle.

UPDATEBOT name battlestatus teamcolor *Source: client*

Description

Sent by client when he is trying to update status of one of his own bots (only bot owner and battle host may update bot).

battlestatus: Similar to that of the normal client's, see MYBATTLESTATUS for more info.

teamcolor: Should be 32-bit signed integer in decimal system (e.g. 255 and not FF) where each color channel should occupy 1 byte (e.g. in hexdecimal: $00BBGGRR, B = blue, G = green, R = red).

UPDATEBOT BATTLE_ID name battlestatus teamcolor *Source: server*

Description

Sent by server notifying client in the battle that one of the bots just got his status updated. Also see UPDATEBOT command.

BATTLE_ID: BATTLE_ID is there just to help client verify that the bot is meant for his battle.

battlestatus: Similar to that of the normal client's, see MYSTATUC for more info.

teamcolor: Should be 32-bit signed integer in decimal system (e.g. 255 and not FF) where each color channel should occupy 1 byte (e.g. in hexdecimal: $00BBGGRR, B = blue, G = green, R = red).

ADDSTARTRECT allyno left top right bottom *Source: client*

Description

Sent by host of the battle adding a start rectangle for 'allyno' ally team. See lobby client implementation and Spring docs for more info on this one. "left", "top", "right" and "bottom" refer to a virtual rectangle that is 200x200 in size, where coordinates should be in interval [0, 200].

ADDSTARTRECT allyno left top right bottom *Source: server*

Description

Sent by server to clients participating in a battle (except for the host). See lobby client implementation and Spring docs for more info on this one. "left", "top", "right" and "bottom" refer to a virtual rectangle that is 200x200 in size, where coordinates should be in interval [0, 200].

REMOVESTARTRECT allyno *Source: client*

Description

Sent by host of the battle removing a start rectangle for 'allyno' ally team. See client implementation and Spring docs for more info on this one.

REMOVESTARTRECT allyno *Source: server*

Description

Sent to clients participating in a battle (except for the host). Also see ADDSTARTRECT command.

SCRIPTSTART *Source: client*

Description

Sent by client who is hosting a battle replay game indicating he is now sending us the game script used in the original replay. Server will then forward this script to all other participants in his battle. Correct sequence of commands when sending the script file is this:

- 1) SCRIPTSTART command
- 2) multiple SCRIPT commands
- 3) SCRIPTEND command

SCRIPTSTART *Source: server*

Description

Sent by server to clients participating in a battle replay indicating that server will now begin sending game script from the original replay file (also see SCRIPTSTART comments when sent by the client).

SCRIPT {line} *Source: client*

Description

Sent by client who previously sent SCRIPTSTART command. Multiple commands of this type are expected after SCRIPTSTART command, ending with SCRIPTEND command as the last command in the sequence (also see SCRIPTSTART comments when sent by the client).

line: This is s a line read from the replay script file (use one SCRIPT command for each line!).

SCRIPT {line} *Source: server*

Description

Sent by server after sending SCRIPTSTART command. Multiple commands of this type are expected after SCRIPTSTART command, ending with SCRIPTEND command as the last command in the sequence (also see comments for SCRIPTSTART command when sent by the client).

line: This is s a line read from the replay script file (one line per one SCRIPT command).

SCRIPTEND *Source: client*

Description

Sent by client indicating he has finished sending us the game script from the battle replay (also see comments for SCRIPTSTART command).

SCRIPTEND *Source: server*

Description

Sent by server indicating he has finished sending the game script from the battle replay (also see comments for SCRIPTSTART command).

SETSCRIPTTAGS {pair1} [{pair2}] [{pair3}] [{...}] *Source: client*

Description

Sent by client (battle host), to set script tags in script.txt. The [pair] format is "key=value can have spaces". Keys may not contain spaces, and are expected to use the '/' character to separate tables (see example). In version 0.35 of TASServer command UPDATEBATTLEDETAILS was completely replaced by this command. The list of attributes that were replaced (with example usage):

- SETSCRIPTTAGS GAME/StartMetal=1000
- SETSCRIPTTAGS GAME/StartEnergy=1000
- SETSCRIPTTAGS GAME/MaxUnits=500
- SETSCRIPTTAGS GAME/StartPosType=1
- SETSCRIPTTAGS GAME/GameMode=0
- SETSCRIPTTAGS GAME/LimitDGun=1
- SETSCRIPTTAGS GAME/DiminishingMMs=0
- SETSCRIPTTAGS GAME/GhostedBuildings=1

Though in reality all tags are joined together in a single SETSCRIPTTAGS command. Note that when specifying multiple key+value pairs, they must be separated by TAB characters. See the examples bellow.

Examples

SETSCRIPTTAGS GAME/MODOPTIONS/TEST=true
SETSCRIPTTAGS GAME/StartMetal=1000 GAME/StartEnergy=1000
See whitespaces: SETSCRIPTTAGS GAME/StartMetal=1000**[TAB]**GAME/StartEnergy=1000

SETSCRIPTTAGS {pair1} [{pair2}] [{pair3}] [{...}] *Source: server*

Description

Relayed from battle host's SETSCRIPTTAGS message. Lobby program must cache and process these tags and apply them accordingly when creating "script.txt" file.

Examples

SETSCRIPTTAGS GAME/MODOPTIONS/TEST=true
SETSCRIPTTAGS GAME/StartMetal=1000 GAME/StartEnergy=1000
See whitespaces: SETSCRIPTTAGS GAME/StartMetal=1000**[TAB]**GAME/StartEnergy=1000

REMOVESCRIPTTAGS key1 [key2] [key3] [...] *Source: client*

Description

Sent by client (battle host), to remove script tags in script.txt.

Examples

REMOVESCRIPTTAGS GAME/MODOPTIONS/TEST1 GAME/MODOPTIONS/TEST2

REMOVESCRIPTTAGS key1 [key2] [key3] [...] *Source: server*

Description

Relayed from battle host's REMOVESCRIPTTAGS message. Lobby program must remove these tags from its cached script tags database.

Examples

REMOVESCRIPTTAGS GAME/MODOPTIONS/TEST1 GAME/MODOPTIONS/TEST2

TESTLOGIN username password *Source: client*

Description

Used to test if username and password are valid. Currently this command is restricted to users with admin access only. It is meant to be used with some external server process to identify user via web login form and create a session cookie based on that (will be used for web interface to the server).

Response

Server will respond with either TESTLOGINACCEPT or TESTLOGINDENY command.

TESTLOGINACCEPT *Source: server*

Description

Used as a response to a TESTLOGIN command in case username/password pair is valid. Use message ID to bind response to a request.

TESTLOGINDENY *Source: server*

Description

Used as a response to a TESTLOGIN command in case username/password pair is not valid. Use message ID to bind response to a request.

ACQUIREUSERID *Source: server*

Description

When client receives this command, he should immediately respond with a USERID command. If client doesn't have a user ID associated yet, he should generate some random ID (unsigned 32 bit integer) and save it permanently as his user ID. He will then provide this user ID each time he logs on to the server via LOGIN command.

USERID userID *Source: client*

Description

Sent as a response to a ACQUIREUSERID command.

userID: Must be an unsigned 32 bit integer, in a hexadecimal form.

Examples

USERID FA23BB4A