BFBC2 PC Remote Administration Protocol

This is the remote-administration protocol used by BFBC2 PC Server R12.

It is work-in-progress; features are first added to the game, and then controlling commands are added to the Remote Administration interface.

Contents

About 2

Low-level protocol 2

- Packet format 2
  - int32 2
  - Word 2
  - Packet 2
- Protocol behaviour 3
- Comments 3

Parameter formats 4

- String 4
- Boolean 4
- HexString 4
- Password 4
- Filename 4
- Clantag 4
- Player name 4
- Team ID 4
- Squad ID 4
- Player subset 5
- Timeout 5
- Id-type 5
- Player info block 5
- Setting context 6

Server events 7

- Summary 7
- Player events 7
- Misc 8

Client commands 9

- Summary 9
- Misc 10

Query 12
Communication 12
Level 13
Manage players 14
Banning 15
Reserved slots 16
Maplist 17
Variables 19

About

This document describes how to communicate with the Remote Administration interface that is present in BFBC2 PC servers. The protocol is bidirectional, and allows clients to send commands to the server as well as the server to send events to clients.

The protocol is designed for machine-readability, not human-readability. It is the basis for all graphical remote administration tools.

Low-level protocol

Packet format

int32

32-bit unsigned integer

	1 byte	bits 7..0 of value
	1 byte	bits 15..8 of value
	1 byte	bits 23..16 of value
	1 byte	bits 31..24 of value

Word

int32	Size	Number of bytes in word, excluding trailing null byte
char[]	Content	Word contents -- must not contain any null bytes
char	Terminator	Trailing null byte

Packet

int32	Sequence	Bit 31: 0 = The command in this command/response pair originated on the server 1 = The command in this command/response pair originated on the client Bit 30: 0 = Request, 1 = Response Bits 29..0: Sequence number (this is used to match requests/responses in a full duplex transmission)
int32	Size	Total size of packet, in bytes

int32	NumWords	Number of words following the packet header
Word[N]	Words	N words

A packet cannot be more than 16384 bytes in size.

Protocol behaviour

The client communicates with the server using a request/response protocol. Each request contains a sequence number which grows monotonically, a flag which indicates whether the command originated on the client or the server, and one word containing the command name. In addition to this, a command can have zero or more arguments.

Every request must be acknowledged by a response. The response includes the the same sequence number, and the same origin flag. However, it has the response flag set.

Sequence numbers are unique within one server-client connection. Thus, the same sequence number can be used when the server is communicating with different clients.

Responses must contain at least one word. The first word can be one of the following:

OK	- request completed successfully
UnknownCommand	- unknown command
InvalidArguments	- Arguments not appropriate for command
<other>	- command-specific error

OK is the only response which signifies success.
Subsequent arguments (if any) are command-specific.

The server is guaranteed to adher to this protocol specification. If the client violates the protocol, the server may close the connection without any prior notice.

Comments

The format of the Words portion of a packet is designed such that it shall be easy to split it into individual words in both C++ and Python. Any numerical arguments are always transferred in string form (not in raw binary form).

The protocol is designed to be fully bidirectional.

Parameter formats

String
An 8bit ASCII string. Must not contain any characters with ASCII code 0.

Boolean
Two possible values:

true
false

HexString
A stream of hexadecimal digits. The stream must always contain an even number of digits. Allowed characters are: 0123456789ABCDEF

Password
A password is from 0 up to 16 characters in length, inclusive. The allowed characters are: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

Filename
A filename is from 1 up to 240 characters in length, inclusive. The allowed characters are: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789._-

Clantag
A clan tag is from 0 to an unknown number of characters in length. At the time of writing, it is unclear which the allowed characters are.

Player name
The “player name” (referred to as “Soldier name” in-game) is the persona name which the player chose when logging in to EA Online. One EA Account can have multiple personas.

A player has a name from 4 to 16 characters in length, inclusive. The allowed characters are:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
_ - & ( ) * + . / : ; < = > ? [ ] ^ { | } ~ <space>

When a player is creating a new persona, it is compared against all other persona names; the new name must be unique. The following characters are ignored during the comparison:

- _ <space>

Team ID

An integer.

Team 0 is neutral. Depending on gamemode, there are up to 16 non-neutral teams, numbered 1..16.

Squad ID
An integer.

Squad 0 is “no squad”. Depending on gamemode, there are up to 16 squads numbered 1..16.
Note that squad ID are local within each team; that is, to uniquely identify a squad you need to specify both a Team ID and a Squad ID.

Player subset

Several commands – such as admin.listPlayers – take a player subset as argument.

A player subset is one of the following:

all	- all players on the server
team <team number: integer>	- all players in the specified team
squad <team number: integer> <squad number: integer>	- all players in the specified team+squad
player <player name: string>	- one specific player

Timeout

Some commands, such as bans, take a timeout as argument.

A timeout is one of the following:

perm	- permanent
round	- until end of round
seconds <number of seconds: integer>	- number of seconds

Id-type

Some commands, such as bans, take an id-type as argument

An id-type is one of the following:

name	- Soldier name
ip	- IP address
guid	- Player guid

Player info block

The standard set of info for a group of players contains a lot of different fields. To reduce the risk of having to do backwards-incompatible changes to the protocol, the player info block includes some formatting information.

<number of parameters>	- number of parameters for each player
N x <parameter type: string>	- the parameter types that will be sent below
<number of players>	- number of players following
M x N x <parameter value>	- all parameter values for player 0, then all parameter values for player 1, etc

Current parameters:

name	string	- player name
guid	GUID	- player GUID, or “” if GUID is not yet known
teamId	Team ID	- player’s current team
squadId	Squad ID	- player’s current squad
kills	integer	- number of kills, as shown in the in-game scoreboard
deaths	integer	- number of deaths, as shown in the in-game scoreboard
score	integer	- score, as shown in the in-game scoreboard
ping	integer	- ping (ms), as shown in the in-game scoreboard

Setting context

To be able to store a setting in flexible way we have created a setting context. The setting can be stored as per level, per game mode or a setting the covers all contexts. This is the syntax for those three context types:

all	The setting will be used on all maps and all game modes
gamemode <game mode>	Where game mode is either RUSH, CONQUEST, SQDM, SQRUSH This setting will override a setting that has context all
level <level name>	Where level name is specified like this: levels/mp_XXX This setting will override a setting that has context gamemode or all

Server events

Most commands require the client to be logged in. Before the client has logged in, only 'login.plainText', 'login.hashed', 'logout', 'version', 'serverInfo' and 'quit' commands are available.

Summary

Command	Description
player.onJoin	Player with name <soldier name> has joined the server
player.onAuthenticated	Player with name <soldier name> has been authenticated + got GUID
player.onLeave	with name <soldier name> has left the server
player.onKill	Player with name <killing soldier name> has killed <killed soldier name>
player.onChat	Chat message has been sent to a group of people
player.onKicked	Player with name <soldier name> has been kicked
punkBuster.onMessage	PunkBuster server has output a message
server.onLoadingLevel	Level is loading
server.onLevelStarted	Level is started
player.onSquadChange	Player might have changed squad
player.onTeamChange	Player might have changed team

Player events

Request: player.onJoin <soldier name: string>
Response: OK
Effect: Player with name <soldier name> has joined the server

Request: player.onAuthenticated <soldier name: string> <player GUID: guid>
Response: OK
Effect: Player with name <soldier name> has been authenticated, and has the given GUID

Request: player.onLeave <soldier name: string>
Response: OK
Effect: Player with name <soldier name> has left the server

Request: player.onKill <killing soldier name: string> <killed soldier name: string>
Response: OK
Effect: Player with name <killing soldier name> has killed <killed soldier name>
Suicide is indicated with the same soldier name for killer and victim.
##RSP Comment: onKill does not specify the weapon used to kill you opponent. This would be really handle to monitor our ranked servers and immediately identify if there is anything suspicious (stat-padding) going on

Request: player.onChat <source soldier name: string> <text: string> <target group: player subset>
Response: OK
Effect: Player with name <source soldier name> (or the server, or the server admin) has sent chat

message <text> to some people

Comment: The chat text is as represented before the profanity filtering
If <source soldier name> is “Server”, then the message was sent from the server rather than from an actual player
If sending to a specific player, and the player doesn’t exist, then the target group will be “player” “”

Request: player.onKicked <soldier name: string> <reason: string>
Response: OK
Effect: Player with name <soldier name> has been kicked

Request: player.onSquadChange <soldier name: player name> <team: Team ID> <squad: Squad ID>
Response: OK
Effect: Player might have changed squad

Request: player.onTeamChange <soldier name: player name> <team: Team ID> <squad: Squad ID>
Response: OK
Effect: Player might have changed team

Misc

Request: punkBuster.onMessage <message: string>
Response: OK
Effect: PunkBuster server has output a message
Comment: The entire message is sent as a raw string. It may contain newlines and whatnot.

Request: server.onLoadingLevel <level name: string> <roundsPlayed: int> <roundsTotal: int>
Response: OK
Effect: Level is loading

Request: server.onLevelStarted
Response: OK
Effect: Level is started

Client commands

Most commands require the client to be logged in. Before the client has logged in, only 'login.plainText', 'login.hashed', 'logout', 'version', 'serverInfo', 'listPlayers' and 'quit' commands are available.

Summary

Command	Description
login.plainText <password>	Attempt to login to game server with password
login.hashed	Retrives the salt, used in the hashed password login process
login.hashed <passwordHard>	Sends a hashed password to the server, in an attempt to log in
logout	Logout from game server
quit	Disconnect from server
version	Reports game server type, and build ID
listPlayers <players>	Return list of a group of players on the server, without GUIDs
eventsEnabled <enabled>	Set whether or not the server will send events to the current connection
help	Report which commands the server knows about
admin.runscript <filename>	Process file, runs script lines one-by-one, aborting processing upon error
punkBuster.pb_sv_command <command>	Send a raw PunkBuster command to the PunkBuster server
serverinfo	Query for brief server info
admin.yell <message, duration, players>	Display a message, very visibly on players' screens
admin.say <message, players>	Send a chat message to a group of players
admin.runNextRound	Switch to next round, without ending current
admin.restartRound	Restart current round
admin.endRound <teamID>	End current round, declaring the specified team as winners
admin.runNextLevel	Alias for admin.runNextRound
admin.restartMap	Alias for admin.restartRound
admin.currentLevel	Return current level name
mapList.nextLevelIndex	Get index of next level to be run
mapList.nextLevelIndex <index>	Set index of next level to be run
admin.supportedMaps <play list>	Retrieve maplist of maps supported in this play list
admin.setPlaylist <name>	Set the play list on the server
admin.getPlaylist	Get the current play list for the server
admin.getPlaylists	Get the play lists for the server
admin.kickPlayer <soldier name, reason>	Kick player <soldier name> from server
admin.listPlayers <players>	Return list of a group of players on the server
admin.movePlayer <name, teamID, squadID, forceKill>	Move a player to another team and squad
admin.killPlayer <name>	Kill a player without scoring effects
banList.load	Load list of banned players/IPs/GUIDs from file
banList.save	Save list of banned players/IPs/GUIDs to file
banList.add <id-type, id, timeout, reason>	Add player/IP/GUID to ban list for a certain amount of time
banList.remove <id-type, id>	Remove player/IP/GUID from ban list
banList.clear	Clears ban list
banList.list [startIndex]	Return part of the list of banned players/IPs/GUIDs
reservedSlots.load	Load list of reserved soldier names from file
reservedSlots.save	Save list of reserved soldier names to file
reservedSlots.addPlayer <name>	Add <name> to list of players who can use the reserved slots
reservedSlots.removePlayer <name>	Remove <name> from list of players who can use the reserved slots
reservedSlots.clear	Clear reserved slots list
reservedSlots.list	Retrieve list of players who can utilize the reserved slots

mapList.load	Load list of map names from file
mapList.save	Save maplist to file
mapList.list [rounds]	Retrieve current maplist
mapList.clear	Clears maplist
mapList.remove <index>	Remove map from list
mapList.append <name, rounds>	Add map with name <name> to end of maplist
mapList.insert <index, name, rounds>	Add map with name at the specified index to the maplist
vars.serverName [name]	Set the server name
vars.adminPassword [password]	Set the admin password for the server
vars.gamePassword [password]	Set the game password for the server
vars.punkBuster [enabled]	Set if the server will use PunkBuster or not
vars.hardCore	Set hardcore mode
vars.ranked	Set ranked or not
vars.rankLimit [rank]	Set the highest rank allowed on to the server
vars.teamBalance [enabled]	Set if the server should autobalance
vars.friendlyFire [enabled]	Set if the server should allow team damage
vars.currentPlayerLimit	Retrieve the current maximum number of players
vars.maxPlayerLimit	Retrieve the server-enforced maximum number of players
vars.playerLimit [nr of players]	Set desired maximum number of players
vars.bannerUrl [url]	Set banner url
vars.serverDescription [description]	Set server description
vars.killCam [enabled]	Set if killcam is enabled
vars.miniMap [enabled]	Set if minimap is enabled
vars.crossHair [enabled]	Set if crosshair for all weapons is enabled
vars.3dSpotting [enabled]	Set if spotted targets are visible in the 3d-world
vars.miniMapSpotting [enabled]	Set if spotted targets are visible on the minimap
vars.thirdPersonVehicleCameras [enabled]	ToDo
vars.teamKillCountForKick [count]	Set number of teamkills allowed during a round
vars.teamKillValueForKick [count]	Set max kill-value allowed for a player before he/she is kicked
vars.teamKillValueIncrease [count]	Set kill-value increase for a teamkill
vars.teamKillValueDecreasePerSecond [count]	Set kill-value decrease per second
vars.idleTimeout [time]	Set idle timeout
vars.profanityFilter [enabled]	Set if profanity filter is enabled
levelVars.set <context> <var name> <value>	Set a level-specific variable in a specific context
levelVars.get <context> <var name>	Get a level-specific variable in a specific context
levelVars.evaluate <var name>	Get a level-specific variable in a specific context
levelVars.clear <context> [var name]	Clear some or all level-specific variables
levelVars.list <context> [var name]	List level-specific variables that match the context & variable name

Misc

Request:	login.plainText <password: string>	
Response:	OK	- Login successful, you are now logged in regardless of prior status
Response:	InvalidPassword	- Login unsuccessful, logged-in status unchanged
Response:	PasswordNotSet	- Login unsuccessful, logged-in status unchanged
Response:	InvalidArguments	
Effect:	Attempt to login to game server with password <password>	
Comments:	If you are connecting to the admin interface over the internet, then use login.hashed instead to avoid having evildoers sniff the admin password	

Request: login.hashed
Response: OK <salt: HexString> - Retrieved salt for the current connection
Response: PasswordNotSet - No password set for server, login impossible
Response: InvalidArguments
Effect: Retrieves the salt, used in the hashed password login process
Comments: This is step 1 in the 2-step hashed password process. When using this people cannot sniff your admin password.

Request: login.hashed <passwordHash: HexString>
Response: OK - Login successful, you are now logged in regardless of prior status
Response: PasswordNotSet - No password set for server, login impossible
Response: InvalidPasswordHash - Login unsuccessful, logged-in status unchanged
Response: InvalidArguments
Effect: Sends a hashed password to the server, in an attempt to log in
Comments: This is step 2 in the 2-step hashed password process. When using this people cannot sniff your admin password.

Request: logout
Response: OK - You are now logged out regardless of prior status
Response: InvalidArguments
Effect: Logout from game server

Request: quit
Response: OK
Response: InvalidArguments
Effect: Disconnect from server

Request: version
Response: OK BFBC2 <version>
Response: InvalidArguments
Effect: Reports game server type, and build ID
Comments: Game server type and build ID uniquely identify the server, and the protocol it is running.

Request: listPlayers <players: player subset>
Response: OK <player info>
Response: InvalidArguments
Effect: Return list of all players on the server, but with zeroed out GUIDs

Request: eventsEnabled [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set whether or not the server will send events to the current connection

Request: help
Response: OK <all commands availble on server, as separate words>
Response: InvalidArguments
Effect: Report which commands the server knows about

Request: admin.runScript <filename: filename>
Response: OK
Response: InvalidArguments
Response: InvalidFileName - The filename specified does not follow filename rules
Response: ScriptError <line> <original error…> - Script failed at line <line>, with the given error
Effect: Process file, executing script lines one-by-one, aborting processing upon error

Request: punkBuster.pb_sv_command <command: string>
Response: OK - Command sent to PunkBuster server module
Response: InvalidArguments
Response: InvalidPbServerCommand - Command does not begin with "pb_sv_"
Effect: Send a raw PunkBuster command to the PunkBuster server
Comment: The entire command is to be sent as a single string. Don't split it into multiple words.

Query

Request: serverInfo
Response: OK <serverName> <current playercount> <max playercount> <current gamemode> <current map> <roundsPlayed> <roundsTotal>

Response: InvalidArguments
Effect: Query for brief server info.
Comments: This command can be performed without being logged in.

Communication

Request: admin.yell <message: string> <duration [in ms]: integer> <players: player subset>
Response: OK
Response: InvalidArguments
Response: TooLongMessage
Response: InvalidDuration
Effect: Display a message, very visibly on players' screens, for a certain amount of time. The duration must be more than 0 and at most 60000 ms. The message must be less than 100 characters long.

Request: admin.say <message: string> <players: player subset>
Response: OK
Response: InvalidArguments
Response: TooLongMessage
Effect: Send a chat message to players. The message must be less than 100 characters long.

Level

Request: admin.runNextRound
Response: OK
Response: InvalidArguments
Effect: Switch to next round
Comments: Always successful

Request: admin.restartRound
Response: OK
Response: InvalidArguments
Effect: Restart the current round

Request: admin.endRound <winner: Team ID>
Response: OK
Response: InvalidArguments
Effect: End the current round, declaring <winner> as the winning team

Request: admin.runNextLevel
Comment: Alias for admin.runNextRound

Request: admin.restartMap
Comment: Alias for admin.restartRound

Request: admin.currentLevel
Response: OK <name>
Response: InvalidArguments
Effect: Return current level name

Request: mapList.nextLevelIndex
Response: OK
Response: InvalidArguments
Effect: Get index of next level to be run

Request: mapList.nextLevelIndex <index: integer>
Response: OK
Response: InvalidArguments
Response: InvalidIndex - Level index not available in server map list
Effect: Set index of next level to be run to <index>

Request: admin.supportedMaps <play list: string>
Response: OK <map names>
Response: InvalidArguments
Response: InvalidPlaylist <play list> - Play list doesn't exist on server
Effect: Retrieve maplist of maps supported in this play list

Request: admin.setPlaylist <name: string>
Response: OK - Play list was changed
Response: InvalidArguments
Response: InvalidPlaylist - Play list doesn't exist. Should be RUSH, CONQUEST, SQDM or SQRUSH.
Effect: Set the play list on the server.
Comments: Will only use maps supported for this play list. So the mapList might be invalid
Delay: Change occurs after end of round

Request: admin.getPlaylist
Response: OK <play list>
Response: InvalidArguments
Effect: Get the current play list for the server

Request: admin.getPlaylists
Response: OK <play lists>
Response: InvalidArguments
Effect: Get the play lists for the server

Manage players

Request: admin.kickPlayer <soldier name: player name> [reason: string]
Response: OK - Player did exist, and got kicked
Response: InvalidArguments
Response: PlayerNotFound - Player name doesn't exist on server
Effect: Kick player <soldier name> from server
Comments: Reason text is optional. Default reason is “Kicked by administrator”.

Request: admin.listPlayers <players: player subset>
Response: OK <player info>
Response: InvalidArguments
Effect: Return list of all players on the server

Request: admin.movePlayer <name: player name> <teamId: Team ID> <squadId: Squad ID> <forceKill: boolean>
Response: OK
Response: InvalidArguments
Response: InvalidTeamId
Response: InvalidSquadId
Response: InvalidPlayerName
Response: InvalidForceKill
Response: PlayerNotDead - Player is alive and forceKill is false
Response: SetTeamFailed
Response: SetSquadFailed
Effect: Move a player to another team and/or squad
Comment: Only works if player is dead. This command will kill player if forceKill is true

Request: admin.killPlayer <name: player name>
Response: OK
Response: InvalidArguments
Response: InvalidPlayerName
Response: SoldierNotAlive
Effect: Kill a player without any stats effect

Banning

Request: banList.load
Response: OK
Response: InvalidArguments
Response: InvalidIdType
Response: InvalidBanType
Response: InvalidTimeStamp - A time stamp could not be read
Response: IncompleteBan - Incomplete ban entry at end of file
Response: AccessError - Could not read from file
Effect: Load list of banned players/IPs/GUIDs from file
Comment: 5 lines (Id-type, id, ban-type, time and reason) are retrieved for every ban in the list.
Entries read before getting InvalidIdType, InvalidBanType, InvalidTimeStamp and IncompleteBan is still loaded.

Request: banList.save
Response: OK
Response: InvalidArguments
Response: AccessError - Could not save to file
Effect: Save list of banned players/IPs/GUIDs to file
Comment: 5 lines (Id-type, id, ban-type, time and reason) are stored for every ban in the list.
Every line break has windows “\r\n” characters.

Request: banList.add <id-type: id-type> <id: string> <timeout: timeout> [reason: string]
Response: OK

Response: InvalidArguments
Response: BanListFull
Effect: Add player to ban list for a certain amount of time
Comments: Adding a new player/IP/GUID ban will replace any previous ban for that player/IP/GUID
timeout can take three forms:
perm - permanent [default]
round - until end of round
seconds <integer> - number of seconds until ban expires
Id-type can be any of these
name – A soldier name
ip – An IP address
guid – A player guid
Id could be either a soldier name, ip address or guid depending on id-type.
Reason is optional and defaults to "Banned by admin"; max length 80 chars.
The ban list can contain at most 100 entries.

Request: banList.remove <id-type: id-type> <id: string>
Response: OK
Response: InvalidArguments
Response: NotFound - Id not found in banlist; banlist unchanged
Effect: Remove player/ip/guid from banlist

Request: banList.clear
Response: OK
Response: InvalidArguments
Effect: Clears ban list

Request: banList.list [startOffset : integer]
Response: OK <player ban entries>
Response: InvalidArguments
Effect: Return a section of the list of banned players/IPs/GUIDs.
Comment: The list starts with a number telling how many bans the call returns.
After that, 5 words (Id-type, id, ban-type, time and reason) are received for every ban in the list.
If no startOffset is supplied, it is assumed to be 0.
At most 100 entries will be returned by the command.
To retrieve the full list, perform several banList.list calls with increasing offset until the server returns 0 entries.
(There is an unsolved synchronization problem hidden there: if a ban expires during this process, then one other entry will be skipped during retrieval. There is no known workaround for this.)

Reserved slots

Request: reservedSlots.load
Response: OK
Response: InvalidArguments

Response:	AccessError	- File not found; internal reserved slots list is now empty
Effect:	Load list of soldier names from file. This is a file with one soldier name per line.	
	If loading succeeds, the reserved slots list will get updated.	
	If loading fails, the reserved slots list will remain unchanged.	

Request:	reservedSlots.save	
Response:	OK	
Response:	InvalidArguments	
Response:	AccessError	- Error while saving
Effect:	Save list of reserved soldier names to file. This is a file with one soldier name per line.	
Comment:	If saving fails, the output file may be unchanged or corrupt.	

Request:	reservedSlots.addPlayer <soldier name: player name>	
Response:	OK	
Response:	InvalidArguments	
Response:	PlayerAlreadyInList	- Player is already in the list; reserved slots list unchanged
Effect:	Add <soldier name> to list of players who can use the reserved slots.	

Request:	reservedSlots.removePlayer <soldier name: player name>	
Response:	OK	
Response:	InvalidArguments	
Response:	PlayerNotInList	- Player does not exist in list; reserved slots list unchanged
Effect:	Remove <soldier name> from list of players who can use the reserved slots.	

Request:	reservedSlots.clear	
Response:	OK	
Response:	InvalidArguments	
Effect:	Clear reserved slots list	

Request:	reservedSlots.list	
Response:	OK <soldier names>	
Response:	InvalidArguments	
Effect:	Retrieve list of players who can utilize the reserved slots	

Maplist

Request:	mapList.load	
Response:	OK	- Maplist loaded
Response:	InvalidArguments	
Response:	AccessError	- File not found, internal maplist is now empty
Response:	InvalidPlaylist	- Play list doesn't exist. Should be RUSH, CONQUEST, SQDM or SQRUSH.
Response:	InvalidMapName <name>	- Map with name <name> doesn't exist in playlist/gamemode
Effect:	Load list of map names from file. This is a file with one map name per line.	

Comments: If loading succeeds, the maplist will get updated.
If loading fails, the maplist will remain unchanged.

Request: mapList.save
Response: OK - Maplist saved
Response: InvalidArguments
Response: AccessError - Error while saving, on-disk maplist file possibly corrupted now
Effect: Save maplist to file. This is a file with one map name per line.
Comments: If saving fails, the output file may be unchanged or corrupt.
Every line break has windows "\r\n" characters.

Request: mapList.list [rounds]
Response: OK <N x map names> - map list, without round info
Response: OK <N x (map name, rounds)> - map list, with round info
Response: InvalidArguments
Effect: Retrieve current maplist (with number of rounds for each if round is specified as option)
Comments: If the user hasn't specified the number of rounds explicitly, the number of rounds will be shown as 0; the default number of rounds is currently 2 but may change in the future

Request: mapList.clear
Response: OK
Response: InvalidArguments
Effect: Clears maplist
Comments: If server attempts to switch level while maplist is cleared, nasty things will happen

Request: mapList.remove <index: integer>
Response: OK - Map removed from list
Response: InvalidArguments
Response: InvalidIndex - Index doesn't exist in server map list
Effect: Remove map from list.

Request: mapList.append <name: string> <rounds: int32>
Response: OK - Map appended to list
Response: InvalidArguments
Response: InvalidMapName - Map doesn't exist on server
Effect: Add map with name <name> to end of maplist
Comment: Remember to specify playlist before adding maps
Rounds is an optional argument. If it isn't specified or 0 it will use game mode default.

Request: mapList.insert <index: integer> <name: string> [rounds: int32]
Response: OK - Map inserted to list
Response: InvalidArguments

Response: InvalidMapName - Map doesn't exist on server or negative index
Effect: Add map with name at the specified index to the maplist
Comment: Rounds is an optional argument. If it isn't specified or 0, game mode default will be used.

Variables

Request: vars.serverName [name: string]
Response: OK - for set operation
Response: OK <name> - for get operation
Response: InvalidArguments
Response: TooLongName - for set operation
Effect: Set server name

Request: vars.adminPassword [password: password]
Response: OK - for set operation
Response: OK <password> - for get operation
Response: InvalidArguments
Response: InvalidPassword - password does not conform to password format rules
Effect: Set the admin password for the server, use it with an empty string("") to reset

Request: vars.gamePassword [password: password]
Response: OK - for set operation
Response: OK <password> - for get operation
Response: InvalidArguments
Response: InvalidPassword - password does not conform to password format rules
Response: InvalidConfig - password can't be set if ranked is enabled
Effect: Set the game password for the server, use it with an empty string("") to reset

Request: vars.punkBuster [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Response: InvalidConfig - punkbuster can't be disabled if ranked is enabled
Response: StartupOnlyCallNotAllowed - this command can only be executed from startup.txt
Effect: Set if the server will use PunkBuster or not

Request: vars.hardCore [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set hardcore mode
Delay: Works after map change

Request: vars.ranked [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response InvalidArguments
Response: StartupOnlyCallNotAllowed - this command can only be executed from startup.txt
Effect: Set ranked or not. If enabled: game password will be removed and punkbuster enabled

Request: vars.rankLimit <rank: integer> **##QA: Says 'OK' but still allow higher ranked players to join**
Response: OK - for set operation
Response: OK <rank: integer> - for get operation
Response: InvalidArguments
Effect: Set the highest rank allowed on to the server (integer value).
Comment: To disable rank limit use -1 as value

Request: vars.teamBalance [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set if the server should autobalance

Request: vars.friendlyFire [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Response: LevelNotLoaded - for set operation
Effect: Set if the server should allow team damage
Delay: Works after round restart
Comment: Not available during level load.

Request: vars.currentPlayerLimit
Response: OK <nr of players: integer> - for get operation
Response: ReadOnly - if you try to send any arguments
Response: InvalidArguments
Effect: Retrieve the current maximum number of players
Comment: This value is computed from all the different player limits in effect at any given moment

Request: vars.maxPlayerLimit
Response: OK <nr of players: integer> - for get operation
Response: ReadOnly - if you try to send any arguments
Response: InvalidArguments
Effect: Retrieve the server-enforced maximum number of players
Comment: Setting the user-defined maximum number of players higher than this has no effect

Request: vars.playerLimit [nr of players: integer]
Response: OK - for set operation
Response: OK <nr of players: integer> - for get operation
Response: InvalidArguments
Response: InvalidNrOfPlayers - Player limit must be in the range 8..32
Effect: Set desired maximum number of players
Comment: The effective maximum number of players is also effected by the server provider, and the game engine

Request: vars.bannerUrl [url: string]
Response: OK - for set operation
Response: OK <url: string> - for get operation
Response: InvalidArguments
Response: TooLongUrl - for set operation
Effect: Set banner url
Comment: The banner url needs to be max 63 characters long
The banner needs to be a 512x64 picture smaller than 127kb
Example: admin.setBannerUrl http://www.example.com/banner.jpg

Request: vars.serverDescription <description: string>
Response: OK - for set operation
Response: OK <description: string> - for get operation
Response: InvalidArguments
Response: TooLongDescription - for set operation
Effect: Set server description
Comment: The description needs to be less than 400 characters long; the character '|' marks end-of-line

Request: vars.killCam [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set if killcam is enabled
Delay: Works after map switch

Request: vars.miniMap [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set if minimap is enabled
Delay: Works after map switch

Request: vars.crossHair [enabled: boolean]
Response: OK - for set operation

Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set if crosshair for all weapons is enabled
Delay: Works after map switch

Request: vars.3dSpotting [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set if spotted targets are visible in the 3d-world
Delay: Works after map switch

Request: vars.miniMapSpotting [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set if spotted targets are visible on the minimap
Delay: Works after map switch

Request: vars.thirdPersonVehideCameras [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: <todo>
Delay: Works after map switch

##QA: Works but is bugged. If you change the setting and someone is in a vehicle in 3rd person view when at end of round, that player will be stuck in 3rd person view even though the setting should only allow 1st person view.

Request: vars.teamKillCountForKick [count: integer]
Response: OK - for set operation
Response: OK <count: integer> - for get operation
Response: InvalidArguments
Effect: Set number of teamkills allowed during one round, before the game kicks the player in question
Set to 0 to disable kill counting
Delay: Instantaneous

Request: vars.teamKillValueForKick [count: integer]
Response: OK - for set operation
Response: OK <count: integer> - for get operation
Response: InvalidArguments
Effect: Set the highest kill-value allowed before a player is kicked for teamkilling

Set to 0 to disable kill value mechanism
Delay: Instantaneous

Request: vars.teamKillValueIncrease [count: integer]
Response: OK - for set operation
Response: OK <count: integer> - for get operation
Response: InvalidArguments
Effect: Set the value of a teamkill (adds to the player's current kill-value)
Delay: Instantaneous

Request: vars.teamKillValueDecreasePerSecond [count: integer]
Response: OK - for set operation
Response: OK <count: integer> - for get operation
Response: InvalidArguments
Effect: Set how much every player's kill-value should decrease per second
Delay: Instantaneous

Request: vars.idleTimeout [time: seconds]
Response: OK - for set operation
Response: OK <time: seconds> - for get operation
Response: InvalidArguments
Effect: Set how many seconds a player can be idle before he/she is kicked from server
Set to 0 to disable idle kick
Delay: Instantaneous

Request: vars.profanityFilter [enabled: boolean]
Response: OK - for set operation
Response: OK <enabled: boolean> - for get operation
Response: InvalidArguments
Effect: Set if all players' chat messages should be sent via a profanity filter on the master servers
Delay: Instantaneous

Request: levelVars.set <context: setting context> <varName: string> <value: (variable specific)>
Response: OK - Game variable successfully set
Response: InvalidArguments
Effect: Set a level-specific variable in a specific context
Comment: This overrides a level setting. When level starts it search after level-specific variables overrides in level context, game mode context or global context. If no overrides are found, level default will be used.

Request: levelVars.get <context: setting context> <varName: string>
Response: OK <value>
Response: NotSet - Level-specific variable override not set in the current context

Response: InvalidArguments
Effect: Get a level-specific variable in a specific context

Request: levelVars.evaluate <varName: string>
Response: OK <value>
Response: NotSet - Level-specific variable override does not apply for current level
Response: InvalidArguments
Effect: Answer the question, “what effect do all the level-specific variables have on the current level?”

Request: levelVars.clear <context: setting context> [varName: string]
Response: OK
Response: InvalidArguments
Effect: Clears one or all level-specific variables in the specified context

Request: levelVars.list <context: setting context> [varName: string]
Response: OK <number of matching entries> <entries>
Response: InvalidArguments
Effect: List all level-specific in the specified context [optionally: that match the given variable name]
Comment: Each returned entry is 4 words: <Context-type> <context> <varName> <value>
<context> can be empty string if not applicable.