ListMethods in Trackmania Forever

Quick

Available methods:

system.listMethods

```
array system.listMethods()
```

Return an array of all available XML-RPC methods on this server.

system.methodSignature

```
array system.methodSignature(string)
```

Given the name of a method, return an array of legal signatures. Each signature is an array of strings. The first item of each signature is the return type, and any others items are parameter types.

system.methodHelp

```
string system.methodHelp(string)
```

Given the name of a method, return a help string.

system.multicall

```
array system.multicall(array)
```

Process an array of calls, and return an array of results. Calls should be structs of the form {'methodName': string, 'params': array}. Each result will either be a single-item array containing the result value, or a struct of the form {'faultCode': int, 'faultString': string}. This is useful when you need to make lots of small calls without lots of round trips.

Authenticate

```
boolean Authenticate(string, string)
```

Allow user authentication by specifying a login and a password, to gain access to the set of functionalities corresponding to this authorization level.

ChangeAuthPassword

```
boolean ChangeAuthPassword(string, string)
```

Change the password for the specified login/user. Only available to SuperAdmin.

EnableCallbacks

```
boolean EnableCallbacks(boolean)
```

Allow the GameServer to call you back.

GetVersion

```
struct GetVersion()
```

Returns a struct with the *Name*, *Version* and *Build* of the application remotely controled.

CallVote

```
boolean CallVote(string)
```

Call a vote for a cmd. The command is a XML string corresponding to an XmlRpc request. Only available to Admin.

CallVoteEx

```
boolean CallVoteEx(string, double, int, int)
```

Extended call vote. Same as CallVote, but you can additionally supply specific parameters for this vote: a ratio, a time out and who is voting. Special timeout values: a timeout of '0' means default, '1' means indefinite; a ratio of '-1' means default; Voters values: '0' means only active players, '1' means any player, '2' is for everybody, pure spectators included. Only available to Admin.

InternalCallVote

```
boolean InternalCallVote()
```

Used internally by game.

CancelVote

```
boolean CancelVote()
```

Cancel the current vote. Only available to Admin.

GetCurrentCallVote

```
struct GetCurrentCallVote()
```

Returns the vote currently in progress. The returned structure is { CallerLogin, CmdName, CmdParam }.

SetCallVoteTimeOut

```
boolean SetCallVoteTimeOut(int)
```

Set a new timeout for waiting for votes. A zero value disables callvote. Only available to Admin. Requires a challenge restart to be taken into account.

GetCallVoteTimeOut

```
struct GetCallVoteTimeOut()
```

Get the current and next timeout for waiting for votes. The struct returned contains two fields 'CurrentValue' and 'NextValue'.

SetCallVoteRatio

```
boolean SetCallVoteRatio(double)
```

Set a new default ratio for passing a vote. Must lie between 0 and 1. Only available to Admin.

GetCallVoteRatio

```
double GetCallVoteRatio()
```

Get the current default ratio for passing a vote. This value lies between 0 and 1.

SetCallVoteRatios

```
boolean SetCallVoteRatios(array)
```

Set new ratios for passing specific votes. The parameter is an array of structs {string *Command*, double *Ratio*}, ratio is in [0,1] or -1 for vote disabled. Only available to Admin.

GetCallVoteRatios

```
array GetCallVoteRatios()
```

Get the current ratios for passing votes.

ChatSendServerMessage

```
boolean ChatSendServerMessage(string)
```

Send a text message to all clients without the server login. Only available to Admin.

ChatSendServerMessageToLanguage

`boolean ChatSendServerMessageToLanguage(array, string)`

Send a localised text message to all clients without the server login, or optionally to a Login (which can be a single login or a list of comma-separated logins). The parameter is an array of structures {*Lang*='??', *Text*='...'}. If no matching language is found, the last text in the array is used. Only available to Admin.

ChatSendServerMessageToId

`boolean ChatSendServerMessageToId(string, int)`

Send a text message without the server login to the client with the specified PlayerId. Only available to Admin.

ChatSendServerMessageToLogin

`boolean ChatSendServerMessageToLogin(string, string)`

Send a text message without the server login to the client with the specified login. Login can be a single login or a list of comma-separated logins. Only available to Admin.

ChatSend

`boolean ChatSend(string)`

Send a text message to all clients. Only available to Admin.

ChatSendToLanguage

`boolean ChatSendToLanguage(array, string)`

Send a localised text message to all clients, or optionally to a Login (which can be a single login or a list of comma-separated logins). The parameter is an array of structures {*Lang*='??', *Text*='...'}. If no matching language is found, the last text in the array is used. Only available to Admin.

ChatSendToLogin

`boolean ChatSendToLogin(string, string)`

Send a text message to the client with the specified login. Login can be a single login or a list of comma-separated logins. Only available to Admin.

ChatSendToId

`boolean ChatSendToId(string, int)`

Send a text message to the client with the specified PlayerId. Only available to Admin.

GetChatLines

`array GetChatLines()`

Returns the last chat lines. Maximum of 40 lines. Only available to Admin.

ChatEnableManualRouting

`boolean ChatEnableManualRouting(boolean, boolean)`

The chat messages are no longer dispatched to the players, they only go to the rpc callback and the controller has to manually forward them. The second (optional) parameter allows all messages from the server to be automatically forwarded. Only available to Admin.

ChatForwardToLogin

`boolean ChatForwardToLogin(string, string, string)`

(Text, SenderLogin, DestLogin) Send a text message to the specified DestLogin (or everybody if empty) on behalf of SenderLogin. DestLogin can be a single login or a list of comma-separated logins. Only available if manual routing is enabled. Only available to Admin.

SendNotice

`boolean SendNotice(string, string, int)`

Display a notice on all clients. The parameters are the text message to display, and the login of the avatar to display next to it (or '' for no avatar), and an optional 'max duration' in seconds (default: 3). Only available to Admin.

SendNoticeToId

`boolean SendNoticeToId(int, string, int, int)`

Display a notice on the client with the specified UId. The parameters are the Uid of the client to whom the notice is sent, the text message to display, and the UId of the avatar to display next to it (or '255' for no avatar), and an optional 'max duration' in seconds (default: 3). Only available to Admin.

SendNoticeToLogin

`boolean SendNoticeToLogin(string, string, string, int)`

Display a notice on the client with the specified login. The parameters are the login of the client to whom the notice is sent, the text message to display, and the login of the avatar to display next to it (or '' for no avatar), and an optional 'max duration' in seconds (default: 3). Login can be a single login or a list of comma-separated logins. Only available to Admin.

SendDisplayManialinkPage

`boolean SendDisplayManialinkPage(string, int, boolean)`

Display a manialink page on all clients. The parameters are the xml description of the page to display, a timeout to autohide it (0 = permanent), and a boolean to indicate whether the page must be hidden as soon as the user clicks on a page option. Only available to Admin.

SendDisplayManialinkPageToId

`boolean SendDisplayManialinkPageToId(int, string, int, boolean)`

Display a manialink page on the client with the specified UId. The first parameter is the UId of the player, the other are identical to 'SendDisplayManialinkPage'. Only available to Admin.

SendDisplayManialinkPageToLogin

`boolean SendDisplayManialinkPageToLogin(string, string, int, boolean)`

Display a manialink page on the client with the specified login. The first parameter is the login of the player, the other are identical to 'SendDisplayManialinkPage'. Login can be a single login or a list of comma-separated logins. Only available to Admin.

SendHideManialinkPage

`boolean SendHideManialinkPage()`

Hide the displayed manialink page on all clients. Only available to Admin.

SendHideManialinkPageToId

`boolean SendHideManialinkPageToId(int)`

Hide the displayed manialink page on the client with the specified UId. Only available to Admin.

SendHideManialinkPageToLogin

`boolean SendHideManialinkPageToLogin(string)`

Hide the displayed manialink page on the client with the specified login. Login can be a single login or a list of comma-separated logins. Only available to Admin.

GetManialinkPageAnswers

`array GetManialinkPageAnswers()`

Returns the latest results from the current manialink page, as an array of structs {string *Login*, int *PlayerId*, int *Result*} Result==0 -> no answer, Result>0.... -> answer from the player.

Kick

`boolean Kick(string, string)`

Kick the player with the specified login, with an optional message. Only available to Admin.

KickId

`boolean KickId(int, string)`

Kick the player with the specified PlayerId, with an optional message. Only available to Admin.

Ban

`boolean Ban(string, string)`

Ban the player with the specified login, with an optional message. Only available to Admin.

BanAndBlackList

`boolean BanAndBlackList(string, string, boolean)`

Ban the player with the specified login, with a message. Add it to the black list, and optionally save the new list. Only available to Admin.

BanId

`boolean BanId(int, string)`

Ban the player with the specified PlayerId, with an optional message. Only available to Admin.

UnBan

`boolean UnBan(string)`

Unban the player with the specified client name. Only available to Admin.

CleanBanList

`boolean CleanBanList()`

Clean the ban list of the server. Only available to Admin.

GetBanList

`array GetBanList(int, int)`

Returns the list of banned players. This method takes two parameters. The first parameter specifies the maximum number of infos to be returned, and the second one the starting index in the list. The list is an array of structures. Each structure contains the following fields : *Login*, *ClientName* and *IPAddress*.

BlackList

`boolean BlackList(string)`

Blacklist the player with the specified login. Only available to SuperAdmin.

BlackListId

`boolean BlackListId(int)`

Blacklist the player with the specified PlayerId. Only available to SuperAdmin.

UnBlackList

`boolean UnBlackList(string)`

UnBlackList the player with the specified login. Only available to SuperAdmin.

CleanBlackList

`boolean CleanBlackList()`

Clean the blacklist of the server. Only available to SuperAdmin.

GetBlackList

`array GetBlackList(int, int)`

Returns the list of blacklisted players. This method takes two parameters. The first parameter specifies the maximum number of infos to be returned, and the second one the starting index in the list. The list is an array of structures. Each structure contains the following fields : *Login*.

LoadBlackList

`boolean LoadBlackList(string)`

Load the black list file with the specified file name. Only available to Admin.

SaveBlackList

`boolean SaveBlackList(string)`

Save the black list in the file with specified file name. Only available to Admin.

AddGuest

`boolean AddGuest(string)`

Add the player with the specified login on the guest list. Only available to Admin.

AddGuestId

`boolean AddGuestId(int)`

Add the player with the specified PlayerId on the guest list. Only available to Admin.

RemoveGuest

`boolean RemoveGuest(string)`

Remove the player with the specified login from the guest list. Only available to Admin.

RemoveGuestId

`boolean RemoveGuestId(int)`

Remove the player with the specified PlayerId from the guest list. Only available to Admin.

CleanGuestList

`boolean CleanGuestList()`

Clean the guest list of the server. Only available to Admin.

GetGuestList

```
array GetGuestList(int, int)
```

Returns the list of players on the guest list. This method takes two parameters. The first parameter specifies the maximum number of infos to be returned, and the second one the starting index in the list. The list is an array of structures. Each structure contains the following fields : *Login*.

LoadGuestList

```
boolean LoadGuestList(string)
```

Load the guest list file with the specified file name. Only available to Admin.

SaveGuestList

```
boolean SaveGuestList(string)
```

Save the guest list in the file with specified file name. Only available to Admin.

SetBuddyNotification

```
boolean SetBuddyNotification(string, boolean)
```

Sets whether buddy notifications should be sent in the chat. *login* is the login of the player, or '' for global setting, and *enabled* is the value. Only available to Admin.

GetBuddyNotification

```
boolean GetBuddyNotification(string)
```

Gets whether buddy notifications are enabled for *login*, or '' to get the global setting.

WriteFile

```
boolean WriteFile(string, base64)
```

Write the data to the specified file. The filename is relative to the Tracks path. Only available to Admin.

TunnelSendDataToId

```
boolean TunnelSendDataToId(int, base64)
```

Send the data to the specified player. Only available to Admin.

TunnelSendDataToLogin

```
boolean TunnelSendDataToLogin(string, base64)
```

Send the data to the specified player. Login can be a single login or a list of comma-separated logins. Only available to Admin.

Echo

```
boolean Echo(string, string)
```

Just log the parameters and invoke a callback. Can be used to talk to other xmlrpc clients connected, or to make custom votes. If used in a callvote, the first parameter will be used as the vote message on the clients. Only available to Admin.

Ignore

```
boolean Ignore(string)
```

Ignore the player with the specified login. Only available to Admin.

IgnoreId

```
boolean IgnoreId(int)
```

Ignore the player with the specified PlayerId. Only available to Admin.

UnIgnore

```
boolean UnIgnore(string)
```

Unignore the player with the specified login. Only available to Admin.

UnIgnoreId

```
boolean UnIgnoreId(int)
```

Unignore the player with the specified PlayerId. Only available to Admin.

CleanIgnoreList

```
boolean CleanIgnoreList()
```

Clean the ignore list of the server. Only available to Admin.

GetIgnoreList

```
array GetIgnoreList(int, int)
```

Returns the list of ignored players. This method takes two parameters. The first parameter specifies the maximum number of infos to be returned, and the second one the starting index in the list. The list is an array of structures. Each structure contains the following fields : *Login*.

Pay

```
int Pay(string, int, string)
```

Pay coppers from the server account to a player, returns the BillId. This method takes three parameters: *Login* of the payee, *Coppers* to pay and a *Label* to send with the payment. The creation of the transaction itself may cost coppers, so you need to have coppers on the server account. Only available to Admin.

SendBill

```
int SendBill(string, int, string, string)
```

Create a bill, send it to a player, and return the BillId. This method takes four parameters: *LoginFrom* of the payer, *Coppers* the player has to pay, *Label* of the transaction and an optional *LoginTo* of the payee (if empty string, then the server account is used). The creation of the transaction itself may cost coppers, so you need to have coppers on the server account. Only available to Admin.

GetBillState

```
struct GetBillState(int)
```

Returns the current state of a bill. This method takes one parameter, the *BillId*. Returns a struct containing *State*, *StateName* and *TransactionId*. Possible enum values are: *CreatingTransaction*, *Issued*, *ValidatingPayement*, *Payed*, *Refused*, *Error*.

GetServerCoppers

```
int GetServerCoppers()
```

Returns the current number of coppers on the server account.

GetSystemInfo

```
struct GetSystemInfo()
```

Get some system infos, including connection rates (in kbps).

SetConnectionRates

```
boolean SetConnectionRates(int, int)
```

Set the download and upload rates (in kbps).

SetServerName

```
boolean SetServerName(string)
```

Set a new server name in utf8 format. Only available to Admin.

GetServerName

```
string GetServerName()
```

Get the server name in utf8 format.

SetServerComment

```
boolean SetServerComment(string)
```

Set a new server comment in utf8 format. Only available to Admin.

GetServerComment

```
string GetServerComment()
```

Get the server comment in utf8 format.

SetHideServer

```
boolean SetHideServer(int)
```

Set whether the server should be hidden from the public server list (0 = visible, 1 = always hidden, 2 = hidden from nations). Only available to Admin.

GetHideServer

```
int GetHideServer()
```

Get whether the server wants to be hidden from the public server list.

IsRelayServer

```
boolean IsRelayServer()
```

Returns true if this is a relay server.

SetServerPassword

```
boolean SetServerPassword(string)
```

Set a new password for the server. Only available to Admin.

GetServerPassword

```
string GetServerPassword()
```

Get the server password if called as Admin or Super Admin, else returns if a password is needed or not.

SetServerPasswordForSpectator

```
boolean SetServerPasswordForSpectator(string)
```

Set a new password for the spectator mode. Only available to Admin.

GetServerPasswordForSpectator

```
string GetServerPasswordForSpectator()
```

Get the password for spectator mode if called as Admin or Super Admin, else returns if a password is needed or not.

SetMaxPlayers

```
boolean SetMaxPlayers(int)
```

Set a new maximum number of players. Only available to Admin. Requires a challenge restart to be taken into account.

GetMaxPlayers

```
struct GetMaxPlayers()
```

Get the current and next maximum number of players allowed on server. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetMaxSpectators

```
boolean SetMaxSpectators(int)
```

Set a new maximum number of Spectators. Only available to Admin. Requires a challenge restart to be taken into account.

GetMaxSpectators

```
struct GetMaxSpectators()
```

Get the current and next maximum number of Spectators allowed on server. The struct returned contains two fields *CurrentValue* and *NextValue*.

EnableP2PUpload

```
boolean EnableP2PUpload(boolean)
```

Enable or disable peer-to-peer upload from server. Only available to Admin.

IsP2PUpload

```
boolean IsP2PUpload()
```

Returns if the peer-to-peer upload from server is enabled.

EnableP2PDownload

```
boolean EnableP2PDownload(boolean)
```

Enable or disable peer-to-peer download for server. Only available to Admin.

IsP2PDownload

```
boolean IsP2PDownload()
```

Returns if the peer-to-peer download for server is enabled.

AllowChallengeDownload

```
boolean AllowChallengeDownload(boolean)
```

Allow clients to download challenges from the server. Only available to Admin.

IsChallengeDownloadAllowed

```
boolean IsChallengeDownloadAllowed()
```

Returns if clients can download challenges from the server.

AutoSaveReplays

```
boolean AutoSaveReplays(boolean)
```

Enable the autosaving of all replays (vizualisable replays with all players, but not validable) on the server. Only available to SuperAdmin.

AutoSaveValidationReplays

`boolean AutoSaveValidationReplays(boolean)`

Enable the autosaving on the server of validation replays, every time a player makes a new time. Only available to SuperAdmin.

IsAutoSaveReplaysEnabled

`boolean IsAutoSaveReplaysEnabled()`

Returns if autosaving of all replays is enabled on the server.

IsAutoSaveValidationReplaysEnabled

`boolean IsAutoSaveValidationReplaysEnabled()`

Returns if autosaving of validation replays is enabled on the server.

SaveCurrentReplay

`boolean SaveCurrentReplay(string)`

Saves the current replay (vizualisable replays with all players, but not validable). Pass a filename, or '' for an automatic filename. Only available to Admin.

SaveBestGhostsReplay

`boolean SaveBestGhostsReplay(string, string)`

Saves a replay with the ghost of all the players' best race. First parameter is the login of the player (or '' for all players), Second parameter is the filename, or '' for an automatic filename. Only available to Admin.

GetValidationReplay

`base64 GetValidationReplay(string)`

Returns a replay containing the data needed to validate the current best time of the player. The parameter is the login of the player.

SetLadderMode

`boolean SetLadderMode(int)`

Set a new ladder mode between ladder disabled (0) and forced (1). Only available to Admin. Requires a challenge restart to be taken into account.

GetLadderMode

`struct GetLadderMode()`

Get the current and next ladder mode on server. The struct returned contains two fields *CurrentValue* and *NextValue*.

GetLadderServerLimits

`struct GetLadderServerLimits()`

Get the ladder points limit for the players allowed on this server. The struct returned contains two fields *LadderServerLimitMin* and *LadderServerLimitMax*.

SetVehicleNetQuality

`boolean SetVehicleNetQuality(int)`

Set the network vehicle quality to Fast (0) or High (1). Only available to Admin. Requires a challenge restart to be taken into account.

GetVehicleNetQuality

```
struct GetVehicleNetQuality()
```

Get the current and next network vehicle quality on server. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetServerOptions

```
boolean SetServerOptions(struct)
```

Set new server options using the struct passed as parameters. This struct must contain the following fields : *Name*, *Comment*, *Password*, *PasswordForSpectator*, *NextMaxPlayers*, *NextMaxSpectators*, *IsP2PUpload*, *IsP2PDownload*, *NextLadderMode*, *NextVehicleNetQuality*, *NextCallVoteTimeOut*, *CallVoteRatio*, *AllowChallengeDownload*, *AutoSaveReplays*, and optionally for forever: *RefereePassword*, *RefereeMode*, *AutoSaveValidationReplays*, *HideServer*, *UseChangingValidationSeed*. Only available to Admin. A change of NextMaxPlayers, NextMaxSpectators, NextLadderMode, NextVehicleNetQuality, NextCallVoteTimeOut or UseChangingValidationSeed requires a challenge restart to be taken into account.

GetServerOptions

```
struct GetServerOptions(int)
```

Optional parameter for compatibility: struct version (0 = united, 1 = forever). Returns a struct containing the server options: *Name*, *Comment*, *Password*, *PasswordForSpectator*, *CurrentMaxPlayers*, *NextMaxPlayers*, *CurrentMaxSpectators*, *NextMaxSpectators*, *IsP2PUpload*, *IsP2PDownload*, *CurrentLadderMode*, *NextLadderMode*, *CurrentVehicleNetQuality*, *NextVehicleNetQuality*, *CurrentCallVoteTimeOut*, *NextCallVoteTimeOut*, *CallVoteRatio*, *AllowChallengeDownload* and *AutoSaveReplays*, and additionally for forever: *RefereePassword*, *RefereeMode*, *AutoSaveValidationReplays*, *HideServer*, *CurrentUseChangingValidationSeed*, *NextUseChangingValidationSeed*.

SetServerPackMask

```
boolean SetServerPackMask(string)
```

Defines the packmask of the server. Can be 'United', 'Nations', 'Sunrise', 'Original', or any of the environment names. (Only challenges matching the packmask will be allowed on the server, so that player connecting to it know what to expect.) Only available when the server is stopped. Only available to Admin.

GetServerPackMask

```
string GetServerPackMask()
```

Get the packmask of the server.

SetForcedMods

```
boolean SetForcedMods(boolean, array)
```

Set the mods to apply on the clients. Parameters: *Override*, if true even the challenges with a mod will be overridden by the server setting; and *Mods*, an array of structures [{*EnvName*, *Url*}, ...]. Requires a challenge restart to be taken into account. Only available to Admin.

GetForcedMods

```
struct GetForcedMods()
```

Get the mods settings.

SetForcedMusic

```
boolean SetForcedMusic(boolean, string)
```

Set the music to play on the clients. Parameters: *Override*, if true even the challenges with a custom music will be overridden by the server setting, and a *UrlOrFileName* for the music. Requires a challenge restart to be taken into account. Only available to Admin.

GetForcedMusic

```
struct GetForcedMusic()
```

Get the music setting.

SetForcedSkins

```
boolean SetForcedSkins(array)
```

Defines a list of remappings for player skins. It expects a list of structs *Orig*, *Name*, *Checksum*, *Url*. Orig is the name of the skin to remap, or '*' for any other. Name, Checksum, Url define the skin to use. (They are optional, you may set value '' for any of those. All 3 null means same as Orig). Will only affect players connecting after the value is set. Only available to Admin.

GetForcedSkins

```
array GetForcedSkins()
```

Get the current forced skins.

GetLastConnectionErrorMessage

```
string GetLastConnectionErrorMessage()
```

Returns the last error message for an internet connection. Only available to Admin.

SetRefereePassword

```
boolean SetRefereePassword(string)
```

Set a new password for the referee mode. Only available to Admin.

GetRefereePassword

```
string GetRefereePassword()
```

Get the password for referee mode if called as Admin or Super Admin, else returns if a password is needed or not.

SetRefereeMode

```
boolean SetRefereeMode(int)
```

Set the referee validation mode. 0 = validate the top3 players, 1 = validate all players. Only available to Admin.

GetRefereeMode

```
int GetRefereeMode()
```

Get the referee validation mode.

SetUseChangingValidationSeed

```
boolean SetUseChangingValidationSeed(boolean)
```

Set whether the game should use a variable validation seed or not. Only available to Admin. Requires a challenge restart to be taken into account.

GetUseChangingValidationSeed

`struct GetUseChangingValidationSeed()`

Get the current and next value of UseChangingValidationSeed. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetWarmUp

`boolean SetWarmUp(boolean)`

Sets whether the server is in warm-up phase or not. Only available to Admin.

GetWarmUp

`boolean GetWarmUp()`

Returns whether the server is in warm-up phase.

ChallengeRestart

`boolean ChallengeRestart()`

Restarts the challenge, with an optional boolean parameter *DontClearCupScores* (only available in cup mode). Only available to Admin.

RestartChallenge

`boolean RestartChallenge()`

Restarts the challenge, with an optional boolean parameter *DontClearCupScores* (only available in cup mode). Only available to Admin.

NextChallenge

`boolean NextChallenge()`

Switch to next challenge, with an optional boolean parameter *DontClearCupScores* (only available in cup mode). Only available to Admin.

StopServer

`boolean StopServer()`

Stop the server. Only available to SuperAdmin.

ForceEndRound

`boolean ForceEndRound()`

In Rounds or Laps mode, force the end of round without waiting for all players to giveup/finish. Only available to Admin.

SetGameInfos

`boolean SetGameInfos(struct)`

Set new game settings using the struct passed as parameters. This struct must contain the following fields : *GameMode*, *ChatTime*, *RoundsPointsLimit*, *RoundsUseNewRules*, *RoundsForcedLaps*, *TimeAttackLimit*, *TimeAttackSynchStartPeriod*, *TeamPointsLimit*, *TeamMaxPoints*, *TeamUseNewRules*, *LapsNbLaps*, *LapsTimeLimit*, *FinishTimeout*, and optionally: *AllWarmUpDuration*, *DisableRespawn*, *ForceShowAllOpponents*, *RoundsPointsLimitNewRules*, *TeamPointsLimitNewRules*, *CupPointsLimit*, *CupRoundsPerChallenge*, *CupNbWinners*, *CupWarmUpDuration*. Only available to Admin. Requires a challenge restart to be taken into account.

GetCurrentGameInfo

```
struct GetCurrentGameInfo(int)
```

Optional parameter for compatibility: struct version (0 = united, 1 = forever). Returns a struct containing the current game settings, ie: *GameMode*, *ChatTime*, *NbChallenge*, *RoundsPointsLimit*, *RoundsUseNewRules*, *RoundsForcedLaps*, *TimeAttackLimit*, *TimeAttackSynchStartPeriod*, *TeamPointsLimit*, *TeamMaxPoints*, *TeamUseNewRules*, *LapsNbLaps*, *LapsTimeLimit*, *FinishTimeout*, and additionally for version 1: *AllWarmUpDuration*, *DisableRespawn*, *ForceShowAllOpponents*, *RoundsPointsLimitNewRules*, *TeamPointsLimitNewRules*, *CupPointsLimit*, *CupRoundsPerChallenge*, *CupNbWinners*, *CupWarmUpDuration*.

GetNextGameInfo

```
struct GetNextGameInfo(int)
```

Optional parameter for compatibility: struct version (0 = united, 1 = forever). Returns a struct containing the game settings for the next challenge, ie: *GameMode*, *ChatTime*, *NbChallenge*, *RoundsPointsLimit*, *RoundsUseNewRules*, *RoundsForcedLaps*, *TimeAttackLimit*, *TimeAttackSynchStartPeriod*, *TeamPointsLimit*, *TeamMaxPoints*, *TeamUseNewRules*, *LapsNbLaps*, *LapsTimeLimit*, *FinishTimeout*, and additionally for version 1: *AllWarmUpDuration*, *DisableRespawn*, *ForceShowAllOpponents*, *RoundsPointsLimitNewRules*, *TeamPointsLimitNewRules*, *CupPointsLimit*, *CupRoundsPerChallenge*, *CupNbWinners*, *CupWarmUpDuration*.

GetGameInfos

```
struct GetGameInfos(int)
```

Optional parameter for compatibility: struct version (0 = united, 1 = forever). Returns a struct containing two other structures, the first containing the current game settings and the second the game settings for next challenge. The first structure is named *CurrentGameInfos* and the second *NextGameInfos*.

SetGameMode

```
boolean SetGameMode(int)
```

Set a new game mode between Rounds (0), TimeAttack (1), Team (2), Laps (3), Stunts (4) and Cup (5). Only available to Admin. Requires a challenge restart to be taken into account.

GetGameMode

```
int GetGameMode()
```

Get the current game mode.

SetChatTime

```
boolean SetChatTime(int)
```

Set a new chat time value in milliseconds (actually 'chat time' is the duration of the end race podium, 0 means no podium displayed.). Only available to Admin.

GetChatTime

```
struct GetChatTime()
```

Get the current and next chat time. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetFinishTimeout

```
boolean SetFinishTimeout(int)
```

Set a new finish timeout (for rounds/laps mode) value in milliseconds. 0 means default. 1 means adaptative to the duration of the challenge. Only available to Admin. Requires a challenge restart to be taken into account.

GetFinishTimeout

```
struct GetFinishTimeout()
```

Get the current and next FinishTimeout. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetAllWarmUpDuration

```
boolean SetAllWarmUpDuration(int)
```

Set whether to enable the automatic warm-up phase in all modes. 0 = no, otherwise it's the duration of the phase, expressed in number of rounds (in rounds/team mode), or in number of times the gold medal time (other modes). Only available to Admin. Requires a challenge restart to be taken into account.

GetAllWarmUpDuration

```
struct GetAllWarmUpDuration()
```

Get whether the automatic warm-up phase is enabled in all modes. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetDisableRespawn

```
boolean SetDisableRespawn(boolean)
```

Set whether to disallow players to respawn. Only available to Admin. Requires a challenge restart to be taken into account.

GetDisableRespawn

```
struct GetDisableRespawn()
```

Get whether players are disallowed to respawn. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetForceShowAllOpponents

```
boolean SetForceShowAllOpponents(int)
```

Set whether to override the players preferences and always display all opponents (0=no override, 1=show all, other value=minimum number of opponents). Only available to Admin. Requires a challenge restart to be taken into account.

GetForceShowAllOpponents

```
struct GetForceShowAllOpponents()
```

Get whether players are forced to show all opponents. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetTimeAttackLimit

```
boolean SetTimeAttackLimit(int)
```

Set a new time limit for time attack mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetTimeAttackLimit

```
struct GetTimeAttackLimit()
```

Get the current and next time limit for time attack mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetTimeAttackSynchStartPeriod

`boolean SetTimeAttackSynchStartPeriod(int)`

Set a new synchronized start period for time attack mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetTimeAttackSynchStartPeriod

`struct GetTimeAttackSynchStartPeriod()`

Get the current and synchronized start period for time attack mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetLapsTimeLimit

`boolean SetLapsTimeLimit(int)`

Set a new time limit for laps mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetLapsTimeLimit

`struct GetLapsTimeLimit()`

Get the current and next time limit for laps mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetNbLaps

`boolean SetNbLaps(int)`

Set a new number of laps for laps mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetNbLaps

`struct GetNbLaps()`

Get the current and next number of laps for laps mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetRoundForcedLaps

`boolean SetRoundForcedLaps(int)`

Set a new number of laps for rounds mode (0 = default, use the number of laps from the challenges, otherwise forces the number of rounds for multilaps challenges). Only available to Admin. Requires a challenge restart to be taken into account.

GetRoundForcedLaps

`struct GetRoundForcedLaps()`

Get the current and next number of laps for rounds mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetRoundPointsLimit

`boolean SetRoundPointsLimit(int)`

Set a new points limit for rounds mode (value set depends on UseNewRulesRound). Only available to Admin. Requires a challenge restart to be taken into account.

GetRoundPointsLimit

```
struct GetRoundPointsLimit()
```

Get the current and next points limit for rounds mode (values returned depend on UseNewRulesRound). The struct returned contains two fields *CurrentValue* and *NextValue*.

SetRoundCustomPoints

```
boolean SetRoundCustomPoints(array, boolean)
```

Set the points used for the scores in rounds mode. *Points* is an array of decreasing integers for the players from the first to last. And you can add an optional boolean to relax the constraint checking on the scores. Only available to Admin.

GetRoundCustomPoints

```
array GetRoundCustomPoints()
```

Gets the points used for the scores in rounds mode.

SetUseNewRulesRound

```
boolean SetUseNewRulesRound(boolean)
```

Set if new rules are used for rounds mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetUseNewRulesRound

```
struct GetUseNewRulesRound()
```

Get if the new rules are used for rounds mode (Current and next values). The struct returned contains two fields *CurrentValue* and *NextValue*.

SetTeamPointsLimit

```
boolean SetTeamPointsLimit(int)
```

Set a new points limit for team mode (value set depends on UseNewRulesTeam). Only available to Admin. Requires a challenge restart to be taken into account.

GetTeamPointsLimit

```
struct GetTeamPointsLimit()
```

Get the current and next points limit for team mode (values returned depend on UseNewRulesTeam). The struct returned contains two fields *CurrentValue* and *NextValue*.

SetMaxPointsTeam

```
boolean SetMaxPointsTeam(int)
```

Set a new number of maximum points per round for team mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetMaxPointsTeam

```
struct GetMaxPointsTeam()
```

Get the current and next number of maximum points per round for team mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetUseNewRulesTeam

```
boolean SetUseNewRulesTeam(boolean)
```

Set if new rules are used for team mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetUseNewRulesTeam

```
struct GetUseNewRulesTeam()
```

Get if the new rules are used for team mode (Current and next values). The struct returned contains two fields *CurrentValue* and *NextValue*.

SetCupPointsLimit

```
boolean SetCupPointsLimit(int)
```

Set the points needed for victory in Cup mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetCupPointsLimit

```
struct GetCupPointsLimit()
```

Get the points needed for victory in Cup mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetCupRoundsPerChallenge

```
boolean SetCupRoundsPerChallenge(int)
```

Sets the number of rounds before going to next challenge in Cup mode. Only available to Admin. Requires a challenge restart to be taken into account.

GetCupRoundsPerChallenge

```
struct GetCupRoundsPerChallenge()
```

Get the number of rounds before going to next challenge in Cup mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetCupWarmUpDuration

```
boolean SetCupWarmUpDuration(int)
```

Set whether to enable the automatic warm-up phase in Cup mode. 0 = no, otherwise it's the duration of the phase, expressed in number of rounds. Only available to Admin. Requires a challenge restart to be taken into account.

GetCupWarmUpDuration

```
struct GetCupWarmUpDuration()
```

Get whether the automatic warm-up phase is enabled in Cup mode. The struct returned contains two fields *CurrentValue* and *NextValue*.

SetCupNbWinners

```
boolean SetCupNbWinners(int)
```

Set the number of winners to determine before the match is considered over. Only available to Admin. Requires a challenge restart to be taken into account.

GetCupNbWinners

```
struct GetCupNbWinners()
```

Get the number of winners to determine before the match is considered over. The struct returned contains two fields *CurrentValue* and *NextValue*.

GetCurrentChallengeIndex

```
int GetCurrentChallengeIndex()
```

Returns the current challenge index in the selection, or -1 if the challenge is no longer in the selection.

GetNextChallengeIndex

```
int GetNextChallengeIndex()
```

Returns the challenge index in the selection that will be played next (unless the current one is restarted...)

SetNextChallengeIndex

```
boolean SetNextChallengeIndex(int)
```

Sets the challenge index in the selection that will be played next (unless the current one is restarted...)

GetCurrentChallengeInfo

```
struct GetCurrentChallengeInfo()
```

Returns a struct containing the infos for the current challenge. The struct contains the following fields : *Name*, *UId*, *FileName*, *Author*, *Environnement*, *Mood*, *BronzeTime*, *SilverTime*, *GoldTime*, *AuthorTime*, *CopperPrice*, *LapRace*, *NbLaps* and *NbCheckpoints*.

GetNextChallengeInfo

```
struct GetNextChallengeInfo()
```

Returns a struct containing the infos for the next challenge. The struct contains the following fields : *Name*, *UId*, *FileName*, *Author*, *Environnement*, *Mood*, *BronzeTime*, *SilverTime*, *GoldTime*, *AuthorTime*, *CopperPrice* and *LapRace*. (*NbLaps* and *NbCheckpoints* are also present but always set to -1)

GetChallengeInfo

```
struct GetChallengeInfo(string)
```

Returns a struct containing the infos for the challenge with the specified filename. The struct contains the following fields : *Name*, *UId*, *FileName*, *Author*, *Environnement*, *Mood*, *BronzeTime*, *SilverTime*, *GoldTime*, *AuthorTime*, *CopperPrice* and *LapRace*. (*NbLaps* and *NbCheckpoints* are also present but always set to -1)

CheckChallengeForCurrentServerParams

```
boolean CheckChallengeForCurrentServerParams(string)
```

Returns a boolean if the challenge with the specified filename matches the current server settings.

GetChallengeList

```
array GetChallengeList(int, int)
```

Returns a list of challenges among the current selection of the server. This method take two parameters. The first parameter specifies the maximum number of infos to be returned, and the second one the starting index in the selection. The list is an array of structures. Each structure contains the following fields : *Name*, *UId*, *FileName*, *Environnement*, *Author*, *GoldTime* and *CopperPrice*.

AddChallenge

```
boolean AddChallenge(string)
```

Add the challenge with the specified filename at the end of the current selection. Only available to Admin.

AddChallengeList

```
int AddChallengeList(array)
```

Add the list of challenges with the specified filenames at the end of the current selection. The list of challenges to add is an array of strings. Only available to Admin.

RemoveChallenge

```
boolean RemoveChallenge(string)
```

Remove the challenge with the specified filename from the current selection. Only available to Admin.

RemoveChallengeList

```
int RemoveChallengeList(array)
```

Remove the list of challenges with the specified filenames from the current selection. The list of challenges to remove is an array of strings. Only available to Admin.

InsertChallenge

```
boolean InsertChallenge(string)
```

Insert the challenge with the specified filename after the current challenge. Only available to Admin.

InsertChallengeList

```
int InsertChallengeList(array)
```

Insert the list of challenges with the specified filenames after the current challenge. The list of challenges to insert is an array of strings. Only available to Admin.

ChooseNextChallenge

```
boolean ChooseNextChallenge(string)
```

Set as next challenge the one with the specified filename, if it is present in the selection. Only available to Admin.

ChooseNextChallengeList

```
int ChooseNextChallengeList(array)
```

Set as next challenges the list of challenges with the specified filenames, if they are present in the selection. The list of challenges to choose is an array of strings. Only available to Admin.

LoadMatchSettings

```
int LoadMatchSettings(string)
```

Set a list of challenges defined in the playlist with the specified filename as the current selection of the server, and load the gameinfos from the same file. Only available to Admin.

AppendPlaylistFromMatchSettings

```
int AppendPlaylistFromMatchSettings(string)
```

Add a list of challenges defined in the playlist with the specified filename at the end of the current selection. Only available to Admin.

SaveMatchSettings

```
int SaveMatchSettings(string)
```

Save the current selection of challenge in the playlist with the specified filename, as well as the current gameinfos. Only available to Admin.

InsertPlaylistFromMatchSettings

```
int InsertPlaylistFromMatchSettings(string)
```

Insert a list of challenges defined in the playlist with the specified filename after the current challenge. Only available to Admin.

GetPlayerList

```
array GetPlayerList(int, int, int)
```

Returns the list of players on the server. This method take two parameters. The first parameter specifies the maximum number of infos to be returned, and the second one the starting index in the list, an optional 3rd parameter is used for compatibility: struct version (0 = united, 1 = forever, 2 = forever, including the servers). The list is an array of PlayerInfo structures. Forever PlayerInfo struct is: Login, NickName, PlayerId, TeamId, SpectatorStatus, LadderRanking, and Flags. LadderRanking is 0 when not in official mode, Flags = ForceSpectator(0,1,2) + IsReferee **10 + IsPodiumReady** 100 + IsUsingStereoscopy **1000 + IsManagedByAnOtherServer** 10000 + IsServer **100000 + HasPlayerSlot** 1000000 SpectatorStatus = Spectator + TemporarySpectator **10 + PureSpectator** 100 + AutoTarget **1000 + CurrentTargetId** 10000

GetPlayerInfo

```
struct GetPlayerInfo(string, int)
```

Returns a struct containing the infos on the player with the specified login, with an optional parameter for compatibility: struct version (0 = united, 1 = forever). The structure is identical to the ones from GetPlayerList. Forever PlayerInfo struct is: Login, NickName, PlayerId, TeamId, SpectatorStatus, LadderRanking, and Flags. LadderRanking is 0 when not in official mode, Flags = ForceSpectator(0,1,2) + IsReferee **10 + IsPodiumReady** 100 + IsUsingStereoscopy **1000 + IsManagedByAnOtherServer** 10000 + IsServer **100000 + HasPlayerSlot** 1000000 SpectatorStatus = Spectator + TemporarySpectator **10 + PureSpectator** 100 + AutoTarget **1000 + CurrentTargetId** 10000

GetDetailedPlayerInfo

```
struct GetDetailedPlayerInfo(string)
```

Returns a struct containing the infos on the player with the specified login. The structure contains the following fields : *Login*, *NickName*, *PlayerId*, *TeamId*, *IPAddress*, *DownloadRate*, *UploadRate*, *Language*, *IsSpectator*, *IsInOfficialMode*, a structure named *Avatar*, an array of structures named *Skins*, a structure named *LadderStats*, *HoursSinceZoneInscription* and *OnlineRights* (0: nations account, 3: united account). Each structure of the array *Skins* contains two fields *Environnement* and a struct *PackDesc*. Each structure *PackDesc*, as well as the struct *Avatar*, contains two fields *FileName* and *Checksum*.

GetMainServerPlayerInfo

```
struct GetMainServerPlayerInfo(int)
```

Returns a struct containing the player infos of the game server (ie: in case of a basic server, itself; in case of a relay server, the main server), with an optional parameter for compatibility: struct version (0 = united, 1 = forever). The structure is identical to the ones from GetPlayerList. Forever PlayerInfo struct is: Login, NickName, PlayerId, TeamId, SpectatorStatus, LadderRanking, and Flags. LadderRanking is 0 when not in official mode, Flags = ForceSpectator(0,1,2) + IsReferee **10 + IsPodiumReady** 100 + IsUsingStereoscopy **1000 + IsManagedByAnOtherServer** 10000 + IsServer **100000 + HasPlayerSlot** 1000000 SpectatorStatus = Spectator + TemporarySpectator **10 + PureSpectator** 100 + AutoTarget **1000 + CurrentTargetId** 10000

GetCurrentRanking

`array GetCurrentRanking(int, int)`

Returns the current rankings for the race in progress. (in team mode, the scores for the two teams are returned. In other modes, it's the individual players' scores) This method take two parameters. The first parameter specifies the maximum number of infos to be returned, and the second one the starting index in the ranking. The ranking returned is a list of structures. Each structure contains the following fields : *Login*, *NickName*, *PlayerId*, *Rank*, *BestTime*, *Score*, *NbrLapsFinished* and *LadderScore*. It also contains an array *BestCheckpoints* that contains the checkpoint times for the best race.

GetCurrentRankingForLogin

`array GetCurrentRankingForLogin(string)`

Returns the current ranking for the race in progressof the player with the specified login (or list of comma-separated logins). The ranking returned is a list of structures, that contains the following fields : *Login*, *NickName*, *PlayerId*, *Rank*, *BestTime*, *Score*, *NbrLapsFinished* and *LadderScore*. It also contains an array *BestCheckpoints* that contains the checkpoint times for the best race.

ForceScores

`boolean ForceScores(array, boolean)`

Force the scores of the current game. Only available in rounds and team mode. You have to pass an array of structs {int *PlayerId*, int *Score*}. And a boolean *SilentMode* - if true, the scores are silently updated (only available for SuperAdmin), allowing an external controller to do its custom counting... Only available to Admin/SuperAdmin.

ForcePlayerTeam

`boolean ForcePlayerTeam(string, int)`

Force the team of the player. Only available in team mode. You have to pass the login and the team number (0 or 1). Only available to Admin.

ForcePlayerTeamId

`boolean ForcePlayerTeamId(int, int)`

Force the team of the player. Only available in team mode. You have to pass the playerid and the team number (0 or 1). Only available to Admin.

ForceSpectator

`boolean ForceSpectator(string, int)`

Force the spectating status of the player. You have to pass the login and the spectator mode (0: user selectable, 1: spectator, 2: player). Only available to Admin.

ForceSpectatorId

`boolean ForceSpectatorId(int, int)`

Force the spectating status of the player. You have to pass the playerid and the spectator mode (0: user selectable, 1: spectator, 2: player). Only available to Admin.

ForceSpectatorTarget

`boolean ForceSpectatorTarget(string, string, int)`

Force spectators to look at a specific player. You have to pass the login of the spectator (or '' for all) and the login of the target (or '' for automatic), and an integer for the camera type to use (-1 = leave unchanged, 0 = replay, 1 = follow, 2 = free). Only available to Admin.

ForceSpectatorTargetId

```
boolean ForceSpectatorTargetId(int, int, int)
```

Force spectators to look at a specific player. You have to pass the id of the spectator (or -1 for all) and the id of the target (or -1 for automatic), and an integer for the camera type to use (-1 = leave unchanged, 0 = replay, 1 = follow, 2 = free). Only available to Admin.

SpectatorReleasePlayerSlot

```
boolean SpectatorReleasePlayerSlot(string)
```

Pass the login of the spectator. A spectator that once was a player keeps his player slot, so that he can go back to race mode. Calling this function frees this slot for another player to connect. Only available to Admin.

SpectatorReleasePlayerSlotId

```
boolean SpectatorReleasePlayerSlotId(int)
```

Pass the playerid of the spectator. A spectator that once was a player keeps his player slot, so that he can go back to race mode. Calling this function frees this slot for another player to connect. Only available to Admin.

ManualFlowControlEnable

```
boolean ManualFlowControlEnable(boolean)
```

Enable control of the game flow: the game will wait for the caller to validate state transitions. Only available to Admin.

ManualFlowControlProceed

```
boolean ManualFlowControlProceed()
```

Allows the game to proceed. Only available to Admin.

ManualFlowControlIsEnabled

```
int ManualFlowControlIsEnabled()
```

Returns whether the manual control of the game flow is enabled. 0 = no, 1 = yes by the xml-rpc client making the call, 2 = yes, by some other xml-rpc client. Only available to Admin.

ManualFlowControlGetCurTransition

```
string ManualFlowControlGetCurTransition()
```

Returns the transition that is currently blocked, or '' if none. (That's exactly the value last received by the callback.) Only available to Admin.

CheckEndMatchCondition

```
string CheckEndMatchCondition()
```

Returns the current match ending condition. Return values are: 'Playing', 'ChangeMap' or 'Finished'.

GetNetworkStats

```
struct GetNetworkStats()
```

Returns a struct containing the networks stats of the server. The structure contains the following fields : *Uptime*, *NbrConnection*, *MeanConnectionTime*, *MeanNbrPlayer*, *RecvNetRate*, *SendNetRate*, *TotalReceivingSize*, *TotalSendingSize* and an array of structures named *PlayerNetInfos*. Each structure of the array PlayerNetInfos contains the following fields : *Login*, *IPAddress*, *LastTransferTime*, *DeltaBetweenTwoLastNetState*, *PacketLossRate*. Only available to SuperAdmin.

StartServerLan

```
boolean StartServerLan()
```

Start a server on lan, using the current configuration. Only available to SuperAdmin.

StartServerInternet

```
boolean StartServerInternet(struct)
```

Start a server on internet using the 'Login' and 'Password' specified in the struct passed as parameters. Only available to SuperAdmin.

GetStatus

```
struct GetStatus()
```

Returns the current status of the server.

QuitGame

```
boolean QuitGame()
```

Quit the application. Only available to SuperAdmin.

GameDataDirectory

```
string GameDataDirectory()
```

Returns the path of the game datas directory. Only available to Admin.

GetTracksDirectory

```
string GetTracksDirectory()
```

Returns the path of the tracks directory. Only available to Admin.

GetSkinsDirectory

```
string GetSkinsDirectory()
```

Returns the path of the skins directory. Only available to Admin.