Legacy Lua API Documentation

Release 2.76

ET:Legacy Team

Jan 27, 2019

Contents

1 Standard libraries **3**

2 Cvars **5**
2.1 lua_modules . . . 5
2.2 lua_allowedmodules . . . 5

3 Commands **7**
3.1 Server commands . . . 7
3.2 Client commands . . . 7

4 Functions **9**
4.1 Modules . . . 9
4.2 Printing . . . 10
4.3 Argument handling . . . 10
4.4 Cvars . . . 11
4.5 Configstrings . . . 11
4.6 Server . . . 12
4.7 Clients . . . 12
4.8 Userinfo . . . 13
4.9 String utility . . . 14
4.10 Filesystem . . . 14
4.11 Indexes . . . 15
4.12 Sound . . . 16
4.13 Miscellaneous . . . 16
4.14 Entities . . . 18
4.15 Shaders . . . 20

5 Callbacks **23**
5.1 qagame execution . . . 23
5.2 Client management . . . 24
5.3 Commands . . . 25
5.4 XP . . . 25
5.5 Miscellaneous . . . 26

6 Fields **27**
6.1 Player fields . . . 27
6.2 Entity fields . . . 30

6.3 Field types . . . 32

7 Constants **35**
7.1 CS constants . . . 35
7.2 MAX constants . . . 37
7.3 WP constants . . . 37
7.4 MOD constants . . . 38
7.5 PW constants . . . 40
7.6 SAY constants . . . 40
7.7 EXEC constants . . . 40
7.8 FS constants . . . 41
7.9 Misc constants . . . 41
7.10 Lua constants . . . 41

8 Miscellaneous **43**
8.1 Configstring . . . 43
8.2 Userinfo . . . 44
8.3 SendServerCommand . . . 44
8.4 Damage bitflags . . . 47
8.5 Skill types . . . 48
8.6 Event types . . . 48

9 Database **51**
9.1 Error handling . . . 51
9.2 Drivers . . . 51
9.3 Environment objects . . . 52
9.4 Connection objects . . . 52
9.5 Cursor objects . . . 53
9.6 SQLite3 extensions . . . 54

10 Sample code **55**
10.1 General . . . 55
10.2 Configstring . . . 57
10.3 Inter Process Communication (IPC) . . . 58
10.4 Database . . . 60

Welcome to the Legacy Lua API's documentation!

The **Legacy mod** is the default mod shipped with ET: Legacy. It supports server-side modifications via the Lua scripting language, with the Legacy Lua API being the interface for communication between them.

The embedded **Lua 5.3** interpreter will load user-defined scripts if present in the *legacy* directory. The Lua API provides an "et" library of function calls that allow access to the server engine, and also provides callbacks so a server side mod may trigger on specific server events.

In some cases values can be returned to Legacy mod, whenever something is intercepted (i.e. a command) and prevented to be further handled. This way, new commands can easily be defined or existing ones can be altered.

Through special functions, it is also possible to alter internal structures or entities (manipulate client XP, set and read cvars, remap shaders, etc.). For example, if a player dies the et_Obituary(victim, killer, meansOfDeath) function is executed, and the Lua API allows you to manipulate and control this information.

Note: Like qagame, Lua modules are unloaded and reloaded on *map_restart* and map changes, which means that all global variables and other information is lost. Persistent data can be stored in cvars, external files or database.

Legacy's Lua API follows mostly the ETPub implementation with partial code of the NoQuarter implemention. The ETPub implementation being built to be compatible with ETPro's Lua, all scripts written in ETPro's documentation should be valid and more or less compatible with Legacy mod's Lua API.

Important: As Legacy uses the newer Lua 5.3, you might want to check the **Incompatibilities with the Previous Version** sections of the Lua 5.1, Lua 5.2, and Lua 5.3 manuals while porting scripts written for other mods.

CHAPTER 1

Standard libraries

The following **standard Lua libraries** are initialized by default and are available to scripts:

- basic
- string
- table
- math
- i/o
- os (available features vary depending on your OS)

Tip: For more information about Lua, check out the Reference Manual, the online edition of the book Programming in Lua or the Lua-users wiki.

CHAPTER 2

Cvars

Cvars control loaded modules.

Note: Changing either cvar will cause all currently loaded modules to quit and be unloaded until the next *map_restart*.

2.1 lua_modules

Space separated list of lua modules for Legacy mod to load. Modules will be run in the order listed.

2.2 lua_allowedmodules

If set, only lua modules with the matching SHA1 signatures listed in this cvar will be allowed to load. If empty, all loaded modules are allowed.

CHAPTER 3

Commands

Commands shows information about currently loaded scripts.

3.1 Server commands

3.1.1 lua_status

Lists all currently loaded Lua modules.

3.1.2 lua_restart

Reinitialises all currently loaded Lua modules.

3.1.3 lua_api

Lists all exported functions and constants available to modders.

3.2 Client commands

3.2.1 lua_status

Lists all currently loaded Lua modules.

Note: Lua mods cannot override this client command.

CHAPTER 4

Functions

Functions allow to manipulate and control the server, or alter internal structures and entities.

4.1 Modules

4.1.1 et.RegisterModname(modname)

Registers a descriptive name for this mod.

- **modname** is the name to register the Lua module.

4.1.2 vmnumber = et.FindSelf()

Returns the assigned Lua VM slot number.

- **vmnumber** is the returned slot number assigned to this Lua VM.

4.1.3 modname, signature = et.FindMod(vmnumber)

Returns the name and SHA1 signature for the mod loaded in a VM slot.

- **vmnumber** is the VM slot number of the Lua module.
- **modname, signature** are the returned registered module's name and SHA-1 signature. Returns **nil, nil** if the VM slot is invalid.

4.1.4 success = et.IPCSend(vmnumber, message)

Sends a message string to the mod in the another VM slot.

- **vmnumber** is the VM slot number of the Lua module to send a message to.

- **message** is the message to sent to the Lua module.

Returns 1 if the message is sent successfully, and 0 if it fails.

Important: The mod receiving message must have an et_IPCReceive() callback.

Note: Data cannot be received and sent back in the same server frame.

4.1.5 et_IPCReceive(vmnumber, message)

Called when another module sends an et.IPCSend() message to this module.

- **vmnumber** is the VM slot number of the sender.
- **message** is the message sent.

Important: The sender module must be loaded earlier in the lua_modules cvar, otherwise the receiver module cannot find it.

Tip: See the Inter Process Communication (IPC) sample code for an example of communication between different loaded Lua modules.

4.2 Printing

4.2.1 et.G_Print(text)

Prints text to the server console.

- **text** is the printed string.

4.2.2 et.G_LogPrint(text)

Prints text to the server console and writes it to the server log.

- **text** is the printed and logged string.

4.3 Argument handling

These functions are to be used within the command callback functions.

4.3.1 args = et.ConcatArgs(index)

Returns all arguments beginning concatenated into a single string.

- **index** is the index of the first argument in the concatenated string.
- **args** is the returned concatenated string.

4.3.2 argcount = et.trap_Argc()

Returns the number of command line arguments in the server command.

- **argcount** is the returned count of arguments.

4.3.3 arg = et.trap_Argv(index)

Returns the contents of the command line argument.

- **index** is the index of the argument to return.
- **arg** is the returned argument.

4.4 Cvars

4.4.1 cvarvalue = et.trap_Cvar_Get(name)

Returns the value of the given cvar.

- **name** is the name of the cvar.
- **cvarvalue** is the returned string containing the value. If there is no cvar with the given name, the returning string has zero length.

4.4.2 et.trap_Cvar_Set(name, cvarvalue)

Sets value to a cvar.

- **name** is the name of the cvar to set.
- **cvarvalue** is the new value for the cvar.

4.5 Configstrings

4.5.1 configstring = et.trap_GetConfigstring(index)

Returns content of the configstring index.

- **index** is the index of the configstring. See et.CS_* constants for possible values.
- **configstring** is the returned string containing the full configstring.

4.5.2 et.trap_SetConfigstring(index, value)

Sets the full configstring.

- **index** is the configstring index. See et.CS_* constants for possible values.
- **value** is the full configstring to set.

4.6 Server

4.6.1 et.trap_SendConsoleCommand(when, command)

Sends command to the server console.

- **when** tells when the command is executed. See et.EXEC_* constants for possible values.
- **command** is the full command to execute.

4.7 Clients

4.7.1 et.trap_SendServerCommand(clientnum, command)

Sends the command command to the client clientnum. If clientnum is *-1*, the command is broadcast to all clients.

Tip: See SendServerCommand() for a detailed example usage of possible commands.

4.7.2 et.trap_DropClient(clientnum, reason, bantime)

Disconnects client from the server.

- **clientnum** is the slot number of the client.
- **reason** is the descriptive reason for the kick which is reported to the client.
- **bantime** is the length of the ban in seconds.

4.7.3 clientnum = et.ClientNumberFromString(string)

Searches for one partial match with player name.

- **string** is a pattern to match against client names.
- **clientnum** is the returned client slot number if one match is found, otherwise **nil** is returned (none or more than one match).

4.7.4 et.G_Say(clientNum, mode, text)

Sends a chat command on behalf of client.

- **clientnum** is the slot number of the client.

- **mode** is the broadcast mode. See et.SAY_* constants.
- **text** is the chat text.

4.7.5 et.MutePlayer(clientnum, duration, reason)

Mutes the specified player.

- **clientnum** is the slot number of the client to mute.
- **duration** is the optional duration of the mute in seconds.
- **reason** is the optional reason of the mute.

4.7.6 et.UnmutePlayer(clientnum)

Unmutes the specified player.

- **clientnum** is the slot number of the client to unmute.

4.8 Userinfo

4.8.1 userinfo = et.trap_GetUserinfo(clientNum)

Returns the userinfo string of a client.

- **clientnum** is the slot number of the client.
- **userinfo** is the returned string of the specified client.

4.8.2 et.trap_SetUserinfo(clientnum, userinfo)

Sets the userinfo string of the client to the specified userinfo.

- **clientnum** is the slot number of the client.
- **userinfo** is the userinfo string that replaces the current userinfo.

Note: The *et.ClientUserinfoChanged()* function must be called after this function for the changes to take effect.

4.8.3 et.ClientUserinfoChanged(clientnum)

Loads the new userinfo string of the client and sets the client settings to match it.

- **clientnum** is the slot number of the client.

4.9 String utility

4.9.1 newinfostring = et.Info_RemoveKey(infostring, key)

Removes a key and its associated value from an infostring.

- **infostring** is the infostring from which to remove the key.
- **key** is the key to remove.
- **newinfostring** is the returned modified infostring without the key.

4.9.2 newinfostring = et.Info_SetValueForKey(infostring, key, value)

Sets a value in an infostring.

- **infostring** is the original infostring.
- **key** is the key to set.
- **value** is the value to set to the key. If empty, the key is removed from the infostring.
- **newinfostring** is the returned modified infostring.

4.9.3 keyvalue = et.Info_ValueForKey(infostring, key)

Returns a value from an infostring.

- **infostring** is the infostring from where to search the key.
- **key** is the key which value is returned.
- **keyvalue** is the returned value from the searched key. If key is not present in the infostring, an empty string is returned.

4.9.4 cleanstring = et.Q_CleanStr(string)

Returns string stripped of all color codes and special characters.

- **string** is the string to clean.
- **cleanstring** is the returned cleaned string.

4.10 Filesystem

4.10.1 fd, len = et.trap_FS_FOpenFile(filename, mode)

Opens a file in the local file system.

- **filename** is the name of the file to open. The file is opened under the current working directory and absolute paths will not work.
- **mode** is the access mode the file is opened. See et.FS_* constants for possible values.
- **fd, len** are returned descriptor of the file and the length of the file. On error, len returns **-1**.

4.10.2 filedata = et.trap_FS_Read(fd, count)

Reads from an open file.

- **fd** is the descriptor of the file to read.
- **count** is the amount of bytes to read.
- **filedata** is the returned value that have the read bytes.

4.10.3 count = et.trap_FS_Write(filedata, count, fd)

Writes at the end of an open file.

- **filedata** is a block of bytes to write.
- **count** is the size of the block to write.
- **fd** is the descriptor of the file.
- **count** is the returned amount of bytes written to the file.

4.10.4 et.trap_FS_FCloseFile(fd)

Closes an opened file.

- **fd** is the descriptor of the opened file.

4.10.5 et.trap_FS_Rename(oldname, newname)

Renames a file in the local file system.

- **oldname** is the name of the file to rename.
- **newname** is the name the old file name is changed to.

4.10.6 filelist = et.trap_FS_GetFileList(dirname, fileextension)

Retrieves list of files from a directory.

- **dirname** is the name of the directory.
- **filextension** is the file extension of file names to retrieve.
- **filelist** is the returned array of file names strings.

4.11 Indexes

4.11.1 soundindex = et.G_SoundIndex(filename)

Returns the index to the searched soundfile.

- **filename** is the sound file name that is searched.
- **soundindex** is the returned string index that includes the filename or 0 if not found.

4.11.2 modelindex = et.G_ModelIndex(filename)

Returns the index to the searched model.

- **filename** is the name that is searched.
- **modelindex** is the returned string index that included the filename or 0 if not found.

4.12 Sound

4.12.1 et.G_globalSound(sound)

Plays a sound to all connected clients.

- **sound** is the name of the sound to play.

4.12.2 et.G_Sound(entnum, soundindex)

Plays a sound originating from position of an entity.

- **entnum** is the number of the entity which position is used as the sound origin.
- **soundindex** is the index of the sound that is played.

4.12.3 et.G_ClientSound(clientnum, soundindex)

Plays a sound originating from a client entity to the team members of that client.

- **clientnum** is the slot number of the connected player.
- **soundindex** is the index to the sound to play.

4.13 Miscellaneous

4.13.1 milliseconds = et.trap_Milliseconds()

Returns level time.

- **milliseconds** is the returned time in milliseconds.

4.13.2 success = et.isBitSet(bit, value)

Checks bit status of a bitmask value.

- **bit** is the checked bit.
- **value** is the bitmask value.

Returns 1 if the bit is set in the bitmask value, and 0 if it is not.

4.13.3 et.G_Damage(target, inflictor, attacker, damage, dflags, mod)

Damages target entity on behalf of the attacker entity.

- **target** is the entity number to damage.
- **inflictor** is the entity number that does the damage.
- **attacker** is the entity number that causes the **inflictor** entity to cause damage to **target**.
- **damage** is the amount of damage to inflict.
- **dflags** is the type of damage to inflict. See Damage bitflags for possible values.
- **mod** is the means of death. See et.MOD_* constants for possible values.

4.13.4 et.G_AddSkillPoints(clientNum, skill, points)

Adds points to the client's skill.

- **clientNum** is the slot number of the client.
- **skill** identifies the skill that the points are added to. See Skill types for possible values.
- **points** is the amount of points to add.

4.13.5 et.G_LoseSkillPoints(clientNum, skill, points)

Removes points to the client's skill.

- **clientNum** is the slot number of the client.
- **skill** identifies the skill that the points are removed from. See Skill types for possible values.
- **points** is the amount of points to remove.

4.13.6 et.G_XP_Set (clientNum , xp, skill, add)

Sets XP of the client.

- **clientNum** is the slot number of the client.
- **xp** is the number of XP points.
- **skill** identifies the skill that the points are added to. See Skill types for possible values.
- **add** sets the XP points if 0, or adds to the existing XP points if 1.

4.13.7 et.G_ResetXP (clientNum)

Resets XP of the client.

- **clientNum** is the slot number of the client.

4.13.8 et.AddWeaponToPlayer(clientNum, weapon, ammo, ammoclip, setcurrent)

Adds a weapon to a client.

- **clientNum** is the slot number of the client.
- **weapon** is the weapon to add. See et.WP_* constants for possible values.
- **ammon** is the number of ammo to add.
- **ammoclip** is the number of ammo clip to add.
- **setcurrent** sets the weapon as current weapon if 1, or does not select it if 0.

Note: Adding a weapon does not automatically add its associated alternate weapon.

4.13.9 et.RemoveWeaponFromPlayer(clientNum, weapon)

Removes a weapon from a client.

- **clientNum** is the slot number of the client.
- **weapon** is the weapon to add. See et.WP_* constants for possible values.

Note: Removing a weapon also removes its associated alternate weapon.

4.14 Entities

4.14.1 entnum = et.G_CreateEntity(params)

Creates a new entity.

- **params** are mapscript parameters.
- **entnum** is the returned number of the new entity.

4.14.2 et.G_DeleteEntity(params)

Deletes an entity.

- **params** are mapscript parameters.

4.14.3 entnum = et.G_TempEntity(origin, event)

Spawns a new temp entity to a location.

- **origin** is the location the temp entity is placed.
- **event** is the event type of the entity. See Event types for possible values.
- **entnum** is the returned the number of the new entity.

4.14.4 et.G_FreeEntity(entnum)

Deletes an entity.

- **entnum** is the entity number.

4.14.5 count = et.G_EntitiesFree()

Calculates all free entities.

- **count** is the returned number of free entities.

Note: Free client entities (slots) are not counted.

4.14.6 et.G_SetEntState(entnum, newstate)

Sets an entity state.

- **entnum** is the entity number.
- **newstate** is the new entity state.

4.14.7 et.trap_LinkEntity(entnum)

Links an entity.

- **entnum** is the entity number to link.

4.14.8 et.trap_UnlinkEntity(entnum)

Unlinks an entity.

- **entnum** is the entity number to unlink.

4.14.9 spawnval = et.G_GetSpawnVar(entnum, key)

Returns a value of a spawnvar.

- **entnum** is the entity number of the target.
- **key** is the key for the value to return. See Entity fields for possible values.
- **spawnval** is the returned spawn value.

4.14.10 et.G_SetSpawnVar(entnum, key, value)

Sets spawn value to an entity.

- **entitynum** is the target entity.
- **key** is the key for the value. See Entity fields for possible values.
- **value** is the new value for the key.

4.14.11 variable = et.gentity_get (entnum, fieldname, arrayindex)

Returns a field value associated with an entity.

- **entnum** is the number of the entity.
- **fieldname** is the name of the field to get. See Fields for possible values.
- **arrayindex**, if present, specifies which element of an array entity field to get.
- **variable** is the returned field value. For NULL entities or clients, **nil** is returned.

Note: *arrayindex* is required when accessing array type fields. Array indexes start at 0.

4.14.12 et.gentity_set(entnum, fieldname, arrayindex, value)

Sets a value in an entity.

- **entnum** is the entity number that is manipulated.
- **fieldname** is the name of the field to manipulate. See Fields for possible values.
- **value** is the new value.
- **arrayindex**, if present, specifies which element of an array entity field to set.

4.14.13 et.G_AddEvent(ent, event, eventparm)

Adds an event to the entity event sequence.

- **ent** is the entity which event sequnce is handled.
- **event** is the event to add.
- **eventparm** is optional parameter for the event.

4.15 Shaders

4.15.1 et.G_ShaderRemap(oldShader, newShader)

Remaps shader.

- **oldShader** is the old shader.
- **newShader** is the new shader.

4.15.2 et.G_ResetRemappedShaders()

Resets remapped shaders.

4.15.3 et.G_ShaderRemapFlush()

Flushes remapped shaders.

4.15.4 et.G_SetGlobalFog(params)

Sets global fog to a specific color and density.

- **params** are mapscript fog parameters.

CHAPTER 5

Callbacks

Callbacks trigger on specific server events.

5.1 qagame execution

5.1.1 et_InitGame(levelTime, randomSeed, restart)

Called when qagame initializes.

- **levelTime** is the current level time in milliseconds.
- **randomSeed** is a number that can be used to seed random number generators.
- **restart** indicates if et_InitGame() is being called due to a *map_restart* (1) or not (0).

5.1.2 et_ShutdownGame(restart)

Called when qagame shuts down.

- **restart** indicates if the shutdown is being called due to a *map_restart* (1) or not (0).

5.1.3 et_RunFrame(levelTime)

Called when qagame runs a server frame.

- **levelTime** is the current level time in milliseconds.

5.1.4 et_Quit()

Called when Legacy unloads the mod.

The mod should close all open filedescriptors and perform all cleanup.

5.2 Client management

5.2.1 rejectreason = et_ClientConnect(clientNum, firstTime, isBot)

Called when a client attempts to connect to the server.

- **clientNum** is the client slot id.
- **firstTime** indicates if this is a new connection (1) or a reconnection (0).
- **isBot** indicates if the client is a bot (1) or not (0).

If the mod accepts the connection, it returns **nil**. Otherwise, the mod should return a string describing the reason the client connection was rejected.

5.2.2 et_ClientDisconnect(clientNum)

Called when a client disconnects.

- **clientNum** is the client slot id.

5.2.3 et_ClientBegin(clientNum)

Called when a client begins (becomes active, and enters the gameworld).

- **clientNum** is the client slot id.

5.2.4 et_ClientUserinfoChanged(clientNum)

Called when a client's Userinfo string has changed.

- **clientNum** is the client slot id.

Note: This only gets called when the players *CS_PLAYERS* config string changes, rather than every time the userinfo changes. This only happens for a subset of userinfo fields.

5.2.5 et_ClientSpawn(clientNum, revived, teamChange, restoreHealth)

Called when a client is spawned.

- **clientNum** is the client slot id.
- **revived** indicates if the client was spawned by being revived (1) or not (0).
- **teamChange** indicates if the client changed team (1) or not (0).
- **restoreHealth** indicates if the player health bar is fully restored (1) or not (0).

5.3 Commands

5.3.1 intercepted = et_ClientCommand(clientNum, command)

Called when a command is received from a client.

- **clientNum** is the client slot id.
- **command** is the command.

Returns 1 if the command was intercepted by the mod, and 0 if the command was ignored and passed through to the server (and other mods in the chain).

Tip: The actual command can be accessed through the argument handling functions, as seen in the Sample Code.

5.3.2 intercepted = et_ConsoleCommand()

Called when a command is entered on the server console.

Returns 1 if the command was intercepted, and 0 if the command was ignored and passed through to the server (and other mods in the chain).

Tip: The actual command can be accessed through the argument handling functions, as seen in the Sample Code.

5.4 XP

5.4.1 et_UpgradeSkill(clientNum, skill)

Called when a client gets a skill upgrade.

- **clientNum** is the client slot.
- **skill** is the skill number.

Returns -1 to override (abort) the qagame function, anything else to "passthrough". Callback may modify skills (or do anything else it wants) during passthrough.

5.4.2 et_SetPlayerSkill(clientNum, skill)

Called when a client skill is set.

- **clientNum** is the client slot.
- **skill** is the skill number.

Returns -1 to override (abort) the qagame function, anything else to "passthrough". Callback may modify skills (or do anything else it wants) during passthrough.

5.5 Miscellaneous

5.5.1 et_Print(text)

Called whenever the server or qagame prints a string to the console.

> **Warning: DO NOT TRUST STRINGS OBTAINED IN THIS WAY!**
>
> Text may contain a player name and their chat message, which makes it very easy to spoof.

5.5.2 et_Obituary(target, attacker, meansOfDeath)

Called whenever a player is killed.

- **target** is the victim.
- **attacker** is the killer.
- **meansOfDeath** is the means of death.

5.5.3 et_Damage(target, attacker, damage, damageFlags, meansOfDeath)

Called whenever a player gets damage.

- **target** is the victim.
- **attacker** is the killer.
- **damage** is the amount of damage.
- **damageFlags** controls how damage is inflicted. See Damage bitflags for possible values.
- **meansOfDeath** is the means of death. See et.MOD_* constants for possible values.

5.5.4 et_SpawnEntitiesFromString()

Called when an entity definition is parsed to spawn gentities.

CHAPTER 6

Fields

Fields are entity parameters supported by et.gentity_get() and et.gentity_set() functions.

6.1 Player fields

Name	Type	Flag	Description
noclip	int	ro	
lastKillTime	int	ro	
saved_persistant	int_array	ro	
lastConstructibleBlockingWarnTime	int	ro	
landmineSpottedTime	int	ro	
lasthurt_client	int	ro	
lasthurt_mod	int	ro	
lasthurt_time	int	ro	
respawnTime	int	ro	
inactivityTime	int	rw	
inactivityWarning	int	rw	
combatState	int	ro	
deathAnimTime	int	ro	
deathTime	int	ro	
disguiseClientNum	int	ro	
medals	int	ro	
acc	float	ro	
hspct	float	ro	
freezed	int	rw	
constructSoundTime	int	ro	
pers.connected	int	ro	
pers.netname	string	noptr	
pers.localClient	int	rw	

Continued on next page

Table 1 – continued from previous page

Name	Type	Flag	Description
pers.initialSpawn	int	rw	
pers.enterTime	int	rw	
pers.connectTime	int	ro	
pers.teamState.state	int	rw	
pers.voteCount	int	rw	
pers.complaints	int	rw	
pers.complaintClient	int	rw	
pers.complaintEndTime	int	rw	
pers.lastReinforceTime	int	rw	
pers.applicationClient	int	rw	
pers.applicationEndTime	int	rw	
pers.invitationClient	int	rw	
pers.invitationEndTime	int	rw	
pers.propositionClient	int	rw	
pers.propositionClient2	int	rw	
pers.propositionEndTime	int	rw	
pers.autofireteamEndTime	int	rw	
pers.autofireteamCreateEndTime	int	rw	
pers.autofireteamJoinEndTime	int	rw	
pers.lastSpawnTime	int	ro	
pers.ready	int	rw	
pers.lastkilled_client	int	ro	
pers.lastrevive_client	int	ro	
pers.lastkiller_client	int	ro	
pers.lastammo_client	int	ro	
pers.lasthealth_client	int	ro	
pers.lastteambleed_client	int	ro	
pers.lastteambleed_dmg	int	ro	
pers.playerStats.hitRegions	int_array	ro	
pers.lastBattleSenseBonusTime	int	ro	
pers.lastHQMineReportTime	int	ro	
pers.maxHealth	int	ro	
pers.playerStats.selfkills	int	ro	
ps.pm_flags	int	ro	
ps.pm_time	int	ro	
ps.eFlags	int	ro	
ps.weapon	int	ro	
ps.weaponstate	int	ro	
ps.stats	int_array	rw	
ps.persistant	int_array	rw	
ps.ping	int	ro	
ps.powerups	int_array	rw	
ps.origin	vec3	rw	
ps.ammo	int_array	rw	
ps.ammoclip	int_array	rw	
ps.classWeaponTime	int	rw	
sess.sessionTeam	int	rw	
sess.spectatorTime	int	rw	
sess.spectatorState	int	rw	

Continued on next page

Table 1 – continued from previous page

Name	Type	Flag	Description
sess.spectatorClient	int	rw	
sess.playerType	int	rw	
sess.playerWeapon	int	rw	
sess.playerWeapon2	int	rw	
sess.spawnObjectiveIndex	int	rw	
sess.latchPlayerType	int	rw	
sess.latchPlayerWeapon	int	rw	
sess.latchPlayerWeapon2	int	rw	
sess.ignoreClients	int_array	rw	
sess.muted	int	rw	
sess.skillpoints	float_array	ro	
sess.startskillpoints	float_array	ro	
sess.startxptotal	float	ro	
sess.skill	int_array	rw	
sess.rank	int	rw	
sess.medals	int_array	rw	
sess.referee	int	rw	
sess.rounds	int	rw	
sess.spec_invite	int	rw	
sess.spec_team	int	rw	
sess.kills	int	rw	
sess.deaths	int	rw	
sess.gibs	int	rw	
sess.self_kills	int	rw	
sess.team_kills	int	rw	
sess.team_gibs	int	rw	
sess.damage_given	int	rw	
sess.damage_received	int	rw	
sess.team_damage_given	int	rw	
sess.team_damage_received	int	rw	
sess.time_axis	int	ro	
sess.time_allies	int	ro	
sess.time_played	int	ro	
sess.mu	float	ro	
sess.sigma	float	ro	
sess.oldmu	float	ro	
sess.oldsigma	float	ro	
sess.uci	int	rw	
sess.aWeaponStats	weaponstat	ro	

Note: All the session *sess.** fields will return *nil* unless the entity is associated with a client slot.

Note: All array variables need to be get or set with an additional parameter.

6.2 Entity fields

Name	Type	Flag	Description
activator	entity	ro	
chain	entity	rw	
classname	string	rw	
clipmask	int	rw	
closespeed	float	rw	
count	int	rw	
count2	int	rw	
damage	int	rw	
deathType	int	rw	
delay	float	rw	
dl_atten	int	rw	
dl_color	vec3	rw	
dl_shader	string	ro	
dl_stylestring	string	ro	
duration	float	rw	
end_size	int	rw	
enemy	entity	rw	
entstate	int	ro	
flags	int	ro	
harc	float	rw	
health	int	rw	
inuse	int	rw	
isProp	int	ro	
item	string	ro	
key	int	rw	
message	string	rw	
methodOfDeath	int	rw	
mg42BaseEnt	int	rw	
missionLevel	int	rw	
model	string	ro	
model2	string	ro	
nextTrain	entity	rw	
noise_index	int	rw	
prevTrain	entity	rw	
props_frame_state	int	ro	
r.absmax	vec3	ro	
r.absmin	vec3	ro	
r.bmodel	int	ro	
r.contents	int	rw	
r.currentAngles	vec3	rw	
r.currentOrigin	vec3	rw	
r.eventTime	int	rw	
r.linked	int	ro	
r.maxs	vec3	rw	
r.mins	vec3	rw	
r.ownerNum	int	rw	
r.singleClient	int	rw	

Continued on next page

Table 2 – continued from previous page

Name	Type	Flag	Description
r.svFlags	int	rw	
r.worldflags	int	ro	
radius	int	rw	
random	float	rw	
rotate	vec3	rw	
s.angles	vec3	rw	
s.angles2	vec3	rw	
s.apos	trajectory	rw	
s.clientNum	int	rw	
s.constantLight	int	rw	
s.density	int	rw	
s.dl_intensity	int	rw	
s.dmgFlags	int	rw	
s.eFlags	int	rw	
s.eType	int	rw	
s.effect1Time	int	rw	
s.effect2Time	int	rw	
s.effect3Time	int	rw	
s.frame	int	rw	
s.groundEntityNum	int	ro	
s.loopSound	int	rw	
s.modelindex	int	rw	
s.modelindex2	int	rw	
s.number	int	ro	
s.onFireEnd	int	rw	
s.onFireStart	int	rw	
s.origin	vec3	rw	
s.origin2	vec3	rw	
s.pos	trajectory	rw	
s.powerups	int	ro	
s.solid	int	rw	
s.teamNum	int	rw	
s.time	int	rw	
s.time2	int	rw	
s.weapon	int	ro	
s.eventParm	int	rw	
scriptName	string	ro	
spawnflags	int	ro	
spawnitem	string	ro	
speed	int	rw	
splashDamage	int	rw	
splashMethodOfDeath	int	rw	
splashRadius	int	rw	
start_size	int	rw	
tagName	string	noptr+ro	
tagParent	entity	rw	
takedamage	int	rw	
tankLink	entity	rw	
target	string	rw	

Continued on next page

Table 2 – continued from previous page

Name	Type	Flag	Description
TargetAngles	vec3	rw	
TargetFlag	int	ro	
targetname	string	ro	
teamchain	entity	rw	
teammaster	entity	rw	
track	string	ro	
varc	float	rw	
wait	float	rw	
waterlevel	int	ro	
watertype	int	ro	

6.3 Field types

6.3.1 int

An integer value.

6.3.2 float

A float value.

6.3.3 string

A string.

6.3.4 array

An array is a list of integer or float values. Individual elements of the array are accessed by passing the desired index in the *arrayindex* argument. Valid array indexes are integers from 0 up to some field specific maximum.

Note: The *arrayindex* argument is required when accessing array type fields, so only one element of an array can be accessed in a given call to the et.gentity_get() and et.gentity_set() functions.

6.3.5 vec3

A vec3_t is a 3-element array of numbers, usually used to store and process coordinates in 3D space. Similarly, in Legacy a vector is an array (table indexed by integers) containing 3 numbers. It can be accessed by:

```
origin = et.gentity_get(entNum, "r.currentOrigin") --a vec3 value
x, y, z = origin[1], origin[2], origin[3]
```

6.3.6 trajectory

A trajectory is returned as a lua table as described below:

```
{
  trDuration = <number>, --- int
  trTime = <number>, -- int
  trType = <number>, -- see below for allowed values
  trBase = <vec3_t>, -- vec3, as described above
  trDelta = <vec3_t> -- also a vec3
}
```

The allowed values for *trType* are as follows:

Name	Description
TR_STATIONARY	
TR_INTERPOLATE	non-parametric, but interpolate between snapshots
TR_LINEAR	
TR_LINEAR_STOP	
TR_LINEAR_STOP_BACK	so reverse movement can be different than forward
TR_SINE	value = base + sin(time / duration) * delta
TR_GRAVITY	
TR_GRAVITY_LOW	
TR_GRAVITY_FLOAT	super low grav with no gravity acceleration (floating feathers/fabric/leaves/...)
TR_GRAVITY_PAUSED	has stopped, but will still do a short trace to see if it should be switched back to TR_GRAVITY
TR_ACCELERATE	
TR_DECCELERATE	
TR_SPLINE	
TR_LINEAR_PATH	

Note: Not all values make sense for all entity types.

6.3.7 entity

Entity numbers are integers from 0 through 1023. Some of the entity numbers have special meanings:

Value	Description
0 - (sv_privateclients - 1)	Reserved for clients who connect with the private slot password
0 - 63	Reserved for clients and also the client number
1022	Worldspawn entity
1023	ENTITYNUM_NONE which is used to indicate no entity when an entity number will be passed over the network

Note: Some other fields not listed as type *entity* may take an entity number value. Examples are *mg42BaseEnt* and *s.number*.

CHAPTER 7

Constants

Constants are strings set on the server.

7.1 CS constants

Name	Value
et.CS_SERVERINFO	0
et.CS_SYSTEMINFO	1
et.CS_MUSIC	2
et.CS_MESSAGE	3
et.CS_MOTD	4
et.CS_WARMUP	5
et.CS_VOTE_TIME	6
et.CS_VOTE_STRING	7
et.CS_VOTE_YES	8
et.CS_VOTE_NO	9
et.CS_GAME_VERSION	10
et.CS_LEVEL_START_TIME	11
et.CS_INTERMISSION	12
et.CS_MULTI_INFO	13
et.CS_MULTI_MAPWINNER	14
et.CS_MULTI_OBJECTIVE	15
et.CS_SCREENFADE	17
et.CS_FOGVARS	18
et.CS_SKYBOXORG	19
et.CS_TARGETEFFECT	20
et.CS_WOLFINFO	21
et.CS_FIRSTBLOOD	22
et.CS_ROUNDSCORES1	23

Continued on next page

Table 1 – continued from previous page

Name	Value
et.CS_ROUNDSCORES2	24
et.CS_MAIN_AXIS_OBJECTIVE	25
et.CS_MAIN_ALLIES_OBJECTIVE	26
et.CS_MUSIC_QUEUE	27
et.CS_SCRIPT_MOVER_NAMES	28
et.CS_CONSTRUCTION_NAMES	29
et.CS_VERSIONINFO	30
et.CS_REINFSEEDS	31
et.CS_SERVERTOGGLES	32
et.CS_GLOBALFOGVARS	33
et.CS_AXIS_MAPS_XP	34
et.CS_ALLIED_MAPS_XP	35
et.CS_INTERMISSION_START_TIME	36
et.CS_ENDGAME_STATS	37
et.CS_CHARGETIMES	38
et.CS_FILTERCAMS	39
et.CS_LEGACYINFO	40
et.CS_SVCVAR	41
et.CS_CONFIGNAME	42
et.CS_TEAMRESTRICTIONS	43
et.CS_UPGRADERANGE	44
et.CS_MODELS	64
et.CS_SOUNDS	320 *(CS_MODELS + MAX_MODELS)*
et.CS_SHADER	576 *(CS_SOUNDS + MAX_SOUNDS)*
et.CS_SHADERSTATE	608 *(CS_SHADERS + MAX_CS_SHADERS)*
et.CS_SKINS	609 *(CS_SHADERSTATE + 1)*
et.CS_CHARACTERS	673 *(CS_SKINS + MAX_CS_SKINS)*
et.CS_PLAYERS	689 *(CS_CHARACTERS + MAX_CHARACTERS)*
et.CS_MULTI_SPAWNTARGETS	753 *(CS_PLAYERS + MAX_CLIENTS)*
et.CS_OID_TRIGGERS	769 *(CS_MULTI_SPAWNTARGETS + MAX_MULTI_SPAWNTARGETS)*
et.CS_OID_DATA	787 *(CS_OID_TRIGGERS + MAX_OID_TRIGGERS)*
et.CS_DLIGHTS	805 *(CS_OID_DATA + MAX_OID_TRIGGERS)*
et.CS_SPLINES	821 *(CS_DLIGHTS + MAX_DLIGHT_CONFIGSTRINGS)*
et.CS_TAGCONNECTS	829 *(CS_SPLINES + MAX_SPLINE_CONFIGSTRINGS)*
et.CS_FIRETEAMS	893 *(CS_TAGCONNECTS + MAX_TAGCONNECTS)*
et.CS_CUSTMOTD	905 *(CS_FIRETEAMS + MAX_FIRETEAMS)*
et.CS_STRINGS	911 *(CS_CUSTMOTD + MAX_MOTDLINES)*
et.CS_MAX	943 *(CS_STRINGS + MAX_CSSTRINGS)*

7.2 MAX constants

Name	Value
et.MAX_CLIENTS	64
et.MAX_MODELS	256
et.MAX_SOUNDS	256
et.MAX_CS_SKINS	64
et.MAX_CSSTRINGS	32
et.MAX_CS_SHADERS	32
et.MAX_SERVER_TAGS	256
et.MAX_TAG_FILES	64
et.MAX_MULTI_SPAWNTARGETS	16
et.MAX_DLIGHT_CONFIGSTRINGS	16
et.MAX_SPLINE_CONFIGSTRINGS	8
et.MAX_OID_TRIGGERS	18
et.MAX_CHARACTERS	16
et.MAX_TAGCONNECTS	64
et.MAX_FIRETEAMS	12
et.MAX_MOTDLINES	6

7.3 WP constants

Name	Value	Description
et.WP_NONE	0	No weapon
et.WP_KNIFE	1	Axis Knife Dagger
et.WP_LUGER	2	Luger
et.WP_MP40	3	MP40
et.WP_GRENADE_LAUNCHER	4	Axis Grenade
et.WP_PANZERFAUST	5	Panzerfaust
et.WP_FLAMETHROWER	6	Flamethrower
et.WP_COLT	7	Colt 1911
et.WP_THOMPSON	8	Thompson
et.WP_GRENADE_PINEAPPLE	9	Allies Grenade
et.WP_STEN	10	Sten
et.WP_MEDIC_SYRINGE	11	Syringe
et.WP_AMMO	12	Ammo pack
et.WP_ARTY	13	Artillery
et.WP_SILENCER	14	Silenced Luger
et.WP_DYNAMITE	15	Dynamite
et.WP_SMOKETRAIL	16	Artillery Initial smoke
et.WP_MAPMORTAR	17	Fixed Mortars
et.VERYBIGEXPLOSION	18	Airstrike Explosion effect
et.WP_MEDKIT	19	Medic pack
et.WP_BINOCULARS	20	Binoculars
et.WP_PLIERS	21	Pliers
et.WP_SMOKE_MARKER	22	Airstrike Marker
et.WP_KAR98	23	Kar98 (Axis Rifle)
et.WP_CARBINE	24	M1 Garand

Continued on next page

Table 2 – continued from previous page

Name	Value	Description
et.WP_GARAND	25	Scoped M1 Garand
et.WP_LANDMINE	26	Landmine
et.WP_SATCHEL	27	Satchel
et.WP_SATCHEL_DET	28	Satchel Detonator
et.WP_SMOKE_BOMB	29	Smoke Grenade
et.WP_MOBILE_MG42	30	Mobile MG42
et.WP_K43	31	K43 (Axis Sniper Rifle)
et.WP_FG42	32	FG42
et.WP_DUMMY_MG42	33	Fixed MG42
et.WP_MORTAR	34	Allies Mortar
et.WP_AKIMBO_COLT	35	Akimbo Colts 1911
et.WP_AKIMBO_LUGER	36	Akimbo Lugers
et.WP_GPG40	37	Kar98 (Grenade Loaded)
et.WP_M7	38	M1 Garand (Grenade Loaded)
et.WP_SILENCED_COLT	39	Silenced Colt 1911
et.WP_GARAND_SCOPE	40	Scoped M1 Garand (Scoped Mode)
et.WP_K43_SCOPE	41	K43 (Scoped Mode)
et.WP_FG42SCOPE	42	FG42 (Scoped Mode)
et.WP_MORTAR_SET	43	Allies Deployed Mortar
et.WP_MEDIC_ADRENALINE	44	Adrenaline
et.WP_AKIMBO_SILENCEDCOLT	45	Akimbo Silenced Colts 1911
et.WP_AKIMBO_SILENCEDLUGER	46	Akimbo Silenced Lugers
et.WP_MOBILE_MG42_SET	47	Deployed Mobile MG42
et.WP_KNIFE_KABAR	48	Allies KA-BAR Knife
et.WP_MOBILE_BROWNING	49	Mobile Browning
et.WP_MOBILE_BROWNING_SET	50	Deployed Mobile Browning
et.WP_MORTAR2	51	Axis Mortar
et.WP_MORTAR2_SET	52	Axis Deployed Mortar
et.WP_BAZOOKA	53	Bazooka
et.WP_MP34	54	MP34
et.WP_NUM_WEAPONS	55	Number of weapons

7.4 MOD constants

Name	Value
et.MOD_UNKNOWN	0
et.MOD_MACHINEGUN	1
et.MOD_BROWNING	2
et.MOD_MG42	3
et.MOD_GRENADE	4
et.MOD_KNIFE	5
et.MOD_LUGER	6
et.MOD_COLT	7
et.MOD_MP40	8
et.MOD_THOMPSON	9
et.MOD_STEN	10
et.MOD_GARAND	11
et.MOD_SILENCER	12

Continued on next page

Table 3 – continued from previous page

Name	Value
et.MOD_FG42	13
et.MOD_FG42SCOPE	14
et.MOD_PANZERFAUST	15
et.MOD_GRENADE_LAUNCHER	16
et.MOD_FLAMETHROWER	17
et.MOD_GRENADE_PINEAPPLE	18
et.MOD_MAPMORTAR	19
et.MOD_MAPMORTAR_SPLASH	20
et.MOD_KICKED	21
et.MOD_DYNAMITE	22
et.MOD_AIRSTRIKE	23
et.MOD_SYRINGE	24
et.MOD_AMMO	25
et.MOD_ARTY	26
et.MOD_WATER	27
et.MOD_SLIME	28
et.MOD_LAVA	29
et.MOD_CRUSH	30
et.MOD_TELEFRAG	31
et.MOD_FALLING	32
et.MOD_SUICIDE	33
et.MOD_TARGET_LASER	34
et.MOD_TRIGGER_HURT	35
et.MOD_EXPLOSIVE	36
et.MOD_CARBINE	37
et.MOD_KAR98	38
et.MOD_GPG40	39
et.MOD_M7	40
et.MOD_LANDMINE	41
et.MOD_SATCHEL	42
et.MOD_SMOKEBOMB	43
et.MOD_MOBILE_MG42	44
et.MOD_SILENCED_COLT	45
et.MOD_GARAND_SCOPE	46
et.MOD_CRUSH_CONSTRUCTION	47
et.MOD_CRUSH_CONSTRUCTIONDEATH	48
et.MOD_CRUSH_CONSTRUCTIONDEATH_NOATTACKER	49
et.MOD_K43	50
et.MOD_K43_SCOPE	51
et.MOD_MORTAR	52
et.MOD_AKIMBO_COLT	53
et.MOD_AKIMBO_LUGER	54
et.MOD_AKIMBO_SILENCEDCOLT	55
et.MOD_AKIMBO_SILENCEDLUGER	56
et.MOD_SMOKEGRENADE	57
et.MOD_SWAP_PLACES	58
et.MOD_SWITCHTEAM	59
et.MOD_SHOVE	60
et.MOD_KNIFE_KABAR	61

Continued on next page

Table 3 – continued from previous page

Name	Value
et.MOD_MOBILE_BROWNING	62
et.MOD_MORTAR2	63
et.MOD_BAZOOKA	64
et.MOD_BACKSTAB	65
et.MOD_MP34	66
et.MOD_NUM_MODS	67

7.5 PW constants

Name	Value	Description
et.PW_NONE	0	No powerup (unused)
et.PW_INVULNERABLE	1	Has spawn shield
et.PW_NOFATIGUE	4	Can sprint
et.PW_REDFLAG	5	Holds Axis objective
et.PW_BLUEFLAG	6	Holds Allied objective
et.PW_OPS_DISGUISED	7	Is disguised
et.PW_OPS_CLASS_1	8	Disguised class helper
et.PW_OPS_CLASS_2	9	Disguised class helper
et.PW_OPS_CLASS_3	10	Disguised class helper
et.PW_ADRENALINE	11	Has adrenaline
et.PW_BLACKOUT	14	Spec blackout
et.PW_MVCLIENTLIST	15	Static MV client info
et.PW_NUM_POWERUPS	16	Number of powerups

7.6 SAY constants

Name	Value	Description
et.SAY_ALL	0	Message will be sent to everyone.
et.SAY_TEAM	1	Message will be sent to the client's team.
et.SAY_BUDDY	2	Message will be sent to the client's fireteam.
et.SAY_TEAMNL	3	Message will be sent to the client's team, without location.

7.7 EXEC constants

Name	Description
et.EXEC_NOW	Executes instantly, don't return until completed.
et.EXEC_INSERT	Insert at current position, but don't run yet.
et.EXEC_APPEND	Append at the end of the command buffer.

7.8 FS constants

Name	Description
et.FS_READ	Opens file in read only mode.
et.FS_WRITE	Opens file in write mode, truncates old file if a file already exists.
et.FS_APPEND	Opens file in write mode at the end of file, old file is not erased if it already exists.
et.FS_APPEND_SYNC	Like et.FS_APPEND, but file buffer is flushed to file on hard drive directly after every write operation.

7.9 Misc constants

Name	Value	Description
et.HOSTARCH	"WIN32", "MACOS" or "UNIX"	Host architecture

7.10 Lua constants

Name	Value	Description
LUA_PATH	./legacy/?.lua; ./legacy/lualibs/?.lua; fs_homepath/fs_game/?.lua; fs_homepath/fs_game/lualibs/?.lua	Ease use of the require function to load scripts
LUA_CPATH	./legacy/lualibs/?.so; fs_homepath/legacy/lualibs/?.so	Ease use of the require function to load libraries
LUA_DIRSEP	/	Directory separator
_VERSION	Lua 5.3	Lua version

CHAPTER 8

Miscellaneous

8.1 Configstring

Configstrings are strings (often in the form of a set of *key\value* pairs) set on the server and automatically sent to each client. They can be accessed with the Configstring and String utility functions.

Tip: A group of related configstrings usually only have a symbolic name for the first value, with a number added to get a particular value. For example, to access a user *CS_PLAYERS* configstring you must use *et.trap_GetConfigstring(et.CS_PLAYERS + slotNumber)*.

See et.CS_* constants for available configstrings.

Here is the detailed content of the user **CS_PLAYERS** configstring:

Key	Value	Description
n	pers.netname	Nickname
t	sess.sessionTeam	Team
c	sess.playerType	Class
lc	sess.latchPlayerType	Latched class
r	sess.rank	Rank
m	medalStr	Medals
s	skillStr	Skills
dn	disguiseClientNum	Disguised covert ops
w	sess.playerWeapon	Weapon
lw	sess.latchPlayerWeapon	Latched primary weapon
sw	sess.latchPlayerWeapon2	Latched secondary weapon
mu	sess.muted	Muted
ref	sess.referee	Referee
u	sess.uci	GeoIP ISO 3166-1 country code

8.2 Userinfo

Userinfo strings are strings set on clients for server processing. They can be accessed with the Userinfo functions.

Key	Example Value	Description
cg_uinfo	12 0 100	Client settings [cg_autoreload/cg_autoactivate/cg_predictitems] [cl_timenudge] [cl_maxpackets]
g_password	none	Server password
cl_guid	0123456789ABCDEF0123456789ABCDEF	GUID
cl_wwwDownload	1	Missing files downloading toggle
name	ETLPlayer	Nickname
rate	2500	Rate setting
snaps	20	Snaps setting
protocol	84	Game protocol
qport	4834	Randomly chosen as startup
challenge	-686256943	Random 31 bit integer
ip	192.168.123.45:27960	IP and port

Note: The userinfo string of bots only includes the *cl_guid*, *name*, *rate*, *snap* and *ip* keys/values.

8.3 SendServerCommand

et.trap_SendServerCommand() is used to send a command from the server to one or more clients.

The first argument is the slot number of the client the command is sent to. If it's equal to **-1**, the command is broadcast to all clients.

The following commands can be issued:

8.3.1 Printing

Print a message to the client's console:

```
"print \"Message\n\""
```

Print a message to the client's annoucement area and console:

```
"cpm \"Message\n\""
```

Print a message to the center of the client's screen:

```
"cp \"Message\n\""
```

Print a message to the client's console and writes it to the statsdump file:

```
"sc \"Message\n\""
```

8.3.2 Chatting

Print a message as a global chat message on behalf of the specified client:

```
"chat ClientNum \"Message\""
```

Print a message as a team chat message on behalf of the specified client:

```
"tchat ClientNum \"Message\" X-Location Y-Location Z-Location"
```

- The **X,Y,Z-Location**'s are optional parameters that represent the client's location.

Print a message as a global chat message via rcon (qsay command):

```
"chat \"Message\""
```

8.3.3 Voice Chat

Send a global voice chat on behalf of the specified client:

```
"vchat VoiceOnly ClientNum 50 Vsay-String Vsay-Number \"Custom-Message\"".
```

- **VoiceOnly** prints a global chat message on behalf of ClientNum if set to **0**, or only play the sound if set to **1**.
- **Vsay-String** is the global voice chat message.
- **Vsay-Number** is the vsay number of Vsay as listed in the .voice files. It is by default random, but can be set by the player by passing parameters to the vsay command (*/vsay <Vsay-Number> <Vsay-String>*).
- **Custom-Message** is by default empty (“”). If set, it prints the message in the chat area.

Send a team voice chat on behalf of the specified client:

```
"vtchat VoiceOnly ClientNum 50 Vsay-String X-Location Y-Location Z-Location Vsay-
↪Number \"Custom-Message\""
```

- **VoiceOnly** prints a team chat message on behalf of ClientNum if set to **0**, or only play the sound if set to **1**.
- **Vsay-String** is the team voice chat message.
- **Vsay-Number** is the vsay number of Vsay as listed in the .voice files. It is by default random, but can be set by the player by passing parameters to the vsay command (*/vsay <Vsay-Number> <Vsay-String>*).
- The **X,Y,Z-Location**'s are optional parameters that represent the client's location.
- **Custom-Message** is by default empty (“”). If set, it prints the message in the chat area.

Send a fireteam voice chat on behalf of the specified client:

```
"vbchat VoiceOnly ClientNum 50 Fireteam-String X-Location Y-Location Z-Location Vsay-
↪Number \"Custom-Message\""
```

- **VoiceOnly** prints a fireteam chat message on behalf of ClientNum if set to **0**, or only play the sound if set to **1**.
- **Fireteam-String** is the fireteam voice chat message.
- **Vsay-Number** is the vsay number of Vsay as listed in the .voice files. It is by default random, but can be set by the player by passing parameters to the vsay command (*/vsay <Vsay-Number> <Vsay-String>*).
- The **X,Y,Z-Location**'s are optional parameters that represent the client's location.
- **Custom-Message** is by default empty (“”). If set, it prints the message in the chat area.

8.3.4 Fireteam

Show a fireteam invitation message to the client:

```
"application Number"
```

- if **Number** is **> -1**, the "Accept ...'s application to join your fireteam?" message is displayed. In this case, **Number** is the ClientNum of the applying client.
- if **Number** is **-1**, the "Your application has been submitted" message is displayed.
- if **Number** is **-2**, the "Your application failed" message is displayed.
- if **Number** is **-3**, the "Your application has been approved" message is displayed.
- if **Number** is **-4**, the "Your application reply has been sent" message is displayed.

Show a fireteam proposition message to the client:

```
"proposition Number Number2"
```

- if **Number** is **> -1**, the "Accept ...'s proposition to invite ... to your fireteam?" message is displayed. In this case, **Number** is the ClientNum of the proposed client, and **Number2** is the ClientNum of the proposing player.
- if **Number** is **-1**, the "Your proposition has been submitted" message is displayed.
- if **Number** is **-2**, the "Your proposition was rejected" message is displayed.
- if **Number** is **-3**, the "Your proposition was accepted" message is displayed.
- if **Number** is **-4**, the "Your proposition reply has been sent" message is displayed.
- **Number2** is an optional parameter only used when **Number > -1**.

Show a fireteam invitation message to the client:

```
"invitation Number"
```

- if **Number** is **> -1**, the "Accept ..'s invitation to join your fireteam?" message is displayed. In this case, **Number** is the ClientNum of the applying client.
- if **Number** is **-1**, the "Your invitation has been submitted" message is displayed.
- if **Number** is **-2**, the "Your invitation rejected" message is displayed.
- if **Number** is **-3**, the "Your invitation was accepted" message is displayed.
- if **Number** is **-4**, the "Your invitation reply has been sent" message is displayed.

8.3.5 Others

Show the complaint vote message to the client:

```
"complaint Number"
```

- if **Number** is **> 1**, the "File complaint against ... for team-killing?" message is displayed. In this case, **Number** is the ClientNum of the teamkilling player.
- if **Number** is **-1**, the "Complaint filed" message is displayed.
- if **Number** is **-2**, the "Complaint dismissed" message is displayed.

Set the client game selected spawnpoint:

```
"setspawnpt Number"
```

- **Number** is the selected spawnpoint.

Disconnect the client with a "Server disconnected" message:

```
"disconnect \"reason\""
```

- **reason** is an optional parameter to show a reason after "Server disconnected".

Note: Use et.trap_DropClient() instead.

Set a client's configstring to a string:

```
"cs Number \"String\""
```

- **String** is the new configstring string.

Note: Use et.trap_SetUserinfo() instead.

Replace any texture:

```
"remapShader \"OldShader\" \"NewShader\" #"
```

- **OldShader** is the old shader.
- **NewShader** is the new shader.
- **#** is the Timeoffset, which currently should be left as 0.

Note: Use et.G_ShaderRemap() instead.

8.4 Damage bitflags

Name	Value	Description
DAMAGE_RADIUS	1	Indirect splash damage
DAMAGE_HALF_KNOCKBACK	2	Do less knockback
DAMAGE_NO_KNOCKBACK	4	Do not affect velocity, just view angles
DAMAGE_NO_PROTECTION	8	Armor, shields, invulnerability, godmode have no effect
DAMAGE_NO_TEAM_PROTECTION	16	(unused)
DAMAGE_DISTANCEFALLOFF	32	Distance falloff

8.5 Skill types

Name	Value	Description
SK_BATTLE_SENSE	0	Battle Sense
SK_EXPLOSIVES_AND_CONSTRUCTION	1	Engineering
SK_FIRST_AID	2	First Aid
SK_SIGNALS	3	Signals
SK_LIGHT_WEAPONS	4	Light Weapons
SK_HEAVY_WEAPONS	5	Heavy Weapons
SK_MILITARY_INTELLIGENCE_AND_SCOPED_WEAPONS	6	Covert Ops

8.6 Event types

Name	Value	Description
EV_NONE	0	
EV_FOOTSTEP	1	
EV_FOOTSTEP_METAL	2	(unused)
EV_FOOTSTEP_WOOD	3	(unused)
EV_FOOTSTEP_GRASS	4	(unused)
EV_FOOTSTEP_GRAVEL	5	(unused)
EV_FOOTSTEP_ROOF	6	(unused)
EV_FOOTSTEP_SNOW	7	(unused)
EV_FOOTSTEP_CARPET	8	(unused)
EV_FOOTSPLASH	9	
EV_FOOTWADE	10	(unused)
EV_SWIM	11	
EV_STEP_4	12	
EV_STEP_8	13	
EV_STEP_12	14	
EV_STEP_16	15	
EV_FALL_SHORT	16	
EV_FALL_MEDIUM	17	
EV_FALL_FAR	18	
EV_FALL_NDIE	19	
EV_FALL_DMG_10	20	
EV_FALL_DMG_15	21	
EV_FALL_DMG_25	22	
EV_FALL_DMG_50	23	
EV_WATER_TOUCH	24	
EV_WATER_LEAVE	25	
EV_WATER_UNDER	26	
EV_WATER_CLEAR	27	
EV_ITEM_PICKUP	28	
EV_ITEM_PICKUP_QUIET	29	
EV_GLOBAL_ITEM_PICKUP	30	
EV_NOAMMO	31	
EV_WEAPONSWITCHED	32	
EV_EMPTYCLIP	33	(unused)

Continued on next page

Table 1 – continued from previous page

Name	Value	Description
EV_FILL_CLIP	34	
EV_MG42_FIXED	35	
EV_WEAP_OVERHEAT	36	
EV_CHANGE_WEAPON	37	
EV_CHANGE_WEAPON_2	38	
EV_FIRE_WEAPON	39	
EV_FIRE_WEAPONB	40	
EV_FIRE_WEAPON_LASTSHOT	41	
EV_NOFIRE_UNDERWATER	42	
EV_FIRE_WEAPON_MG42	43	
EV_FIRE_WEAPON_MOUNTEDMG42	44	
EV_ITEM_RESPAWN	45	(unused)
EV_ITEM_POP	46	(unused)
EV_PLAYER_TELEPORT_IN	47	(unused)
EV_PLAYER_TELEPORT_OUT	48	(unused)
EV_GRENADE_BOUNCE	49	
EV_GENERAL_SOUND	50	
EV_GENERAL_SOUND_VOLUME	51	
EV_GLOBAL_SOUND	52	
EV_GLOBAL_CLIENT_SOUND	53	
EV_GLOBAL_TEAM_SOUND	54	
EV_FX_SOUND	55	
EV_BULLET_HIT_FLESH	56	
EV_BULLET_HIT_WALL	57	
EV_MISSILE_HIT	58	
EV_MISSILE_MISS	59	
EV_RAILTRAIL	60	
EV_BULLET	61	
EV_LOSE_HAT	62	
EV_PAIN	63	
EV_CROUCH_PAIN	64	(unused)
EV_DEATH1	65	(unused)
EV_DEATH2	66	(unused)
EV_DEATH3	67	(unused)
EV_OBITUARY	68	
EV_STOPSTREAMINGSOUND	69	
EV_POWERUP_QUAD	70	
EV_POWERUP_BATTLESUIT	71	
EV_POWERUP_REGEN	72	
EV_GIB_PLAYER	73	
EV_DEBUG_LINE,	74	(unused)
EV_STOPLOOPINGSOUND	75	
EV_TAUNT	76	(unused)
EV_SMOKE	77	
EV_SPARKS	78	
EV_SPARKS_ELECTRIC	79	
EV_EXPLODE	80	
EV_RUBBLE	81	
EV_EFFECT	82	

Continued on next page

Table 1 – continued from previous page

Name	Value	Description
EV_MORTAREFX	83	
EV_SPINUP	84	
EV_SNOW_ON	85	(unused)
EV_SNOW_OFF	86	(unused)
EV_MISSILE_MISS_SMALL	87	
EV_MISSILE_MISS_LARGE	88	
EV_MORTAR_IMPACT	89	
EV_MORTAR_MISS	90	
EV_SPIT_HIT	91	(unused)
EV_SPIT_MISS	92	(unused)
EV_SHARD	93	
EV_JUNK	94	
EV_EMITTER	95	
EV_OILPARTICLES	96	
EV_OILSLICK	97	
EV_OILSLICKREMOVE	98	
EV_MG42EFX	99	(unused)
EV_FLAKGUN1	100	(unused)
EV_FLAKGUN2	101	(unused)
EV_FLAKGUN3	102	(unused)
EV_FLAKGUN4	103	(unused)
EV_EXERT1	104	(unused)
EV_EXERT2	105	(unused)
EV_EXERT3	106	(unused)
EV_SNOWFLURRY	107	
EV_CONCUSSIVE	108	(unused)
EV_DUST	109	
EV_RUMBLE_EFX	110	
EV_GUNSPARKS	111	
EV_FLAMETHROWER_EFFECT	112	
EV_POPUP	113	(unused)
EV_POPUPBOOK	114	(unused)
EV_GIVEPAGE	115	(unused)
EV_MG42BULLET_HIT_FLESH	116	
EV_MG42BULLET_HIT_WALL	117	
EV_SHAKE	118	
EV_DISGUISE_SOUND	119	
EV_BUILDDECAYED_SOUND	120	
EV_FIRE_WEAPON_AAGUN	121	
EV_DEBRIS	122	
EV_ALERT_SPEAKER	123	
EV_POPUPMESSAGE	124	
EV_ARTYMESSAGE	125	
EV_AIRSTRIKEMESSAGE	126	
EV_MEDIC_CALL	127	
EV_SHOVE_SOUND	128	
EV_BODY_DP	129	

CHAPTER 9

Database

Both the Legacy mod and the ET:Legacy engine are built-in with SQLite3 database support. The engine allows to execute SQL statement directly in console, while the Legacy mod has access through LuaSQL.

Tip: See the Database sample code for an example of basic database usage.

9.1 Error handling

LuaSQL is just an abstraction layer that communicates between Lua and a database system. Therefore errors can occur on both levels, that is, inside the database client or inside LuaSQL driver.

Errors such as malformed SQL statements, unknown table names etc. are called database errors and will be reported by the function/method returning *nil* followed by the error message provided by the database system. Errors such as wrong parameters, absent connection, invalid objects etc., called API errors, are usually program errors and so will raise a Lua error.

This behavior will be followed by all functions/methods described in this document unless otherwise stated.

9.2 Drivers

A LuaSQL driver allows the use of the LuaSQL API with a database management system that corresponds to the driver. To use a driver you have to load it. The example below:

```
local driver = require "luasql.sqlite3"
```

loads the SQLite3 driver and returns a table with an entry with the name of the driver (sqlite3 in this case).

9.3 Environment objects

An environment object is created by calling the driver's initialization function that is stored in the table returned when it was loaded, indexed with the same name as the driver (odbc, postgres etc). The following example, will try to create an environment object using the SQLite3 driver:

```
local driver = require "luasql.sqlite3"
local env = driver.sqlite3()
```

9.3.1 env:close()

Closes the environment *env*. Only successful if all connections pertaining to it were closed first.

Returns **true** in case of success; **false** when the object is already closed.

9.3.2 env:connect(sourcename[,username[,password]])

Connects to a data source specified in sourcename using username and password if they are supplied. The sourcename may vary according to each driver. SQLite3 uses a simple database name:

```
-- get database path
local dbpath = string.gsub(et.trap_Cvar_Get("fs_homepath"), "\\", "/")..."/"..et.trap_
↪Cvar_Get("fs_game")..."/"
-- connection
conn = assert(env:connect(dbpath .. "etl.db"))
```

Returns a connection object.

If desired, Lua scripts can also connect to the engine memory database by the following Lua command:

```
conn = assert(env:connect("file::memory:?cache=shared"))
```

Note: The database is only active when the db_mode cvar is set.

To save this in memory database to disk use the **saveDB** console command. See also the db_url cvar to specify the database path.

9.4 Connection objects

A connection object contains specific attributes and parameters of a single data source connection. A connection object is created by calling the environment:connect method.

9.4.1 conn:close()

Closes the connection conn. Only successful if all cursors pertaining to it have been closed and the connection is still open.

Returns **true** in case of success and **false** in case of failure.

9.4.2 conn:commit()

Commits the current transaction.

Returns **true** in case of success and **false** when the operation could not be performed or when it is not implemented.

9.4.3 conn:execute(statement)

Executes the given *SQL statement*.

Returns a cursor object if there are results, or the **number of rows** affected by the command otherwise.

9.4.4 conn:rollback()

Rolls back the current transaction.

Returns **true** in case of success and **false** when the operation could not be performed or when it is not implemented.

9.4.5 conn:setautocommit(boolean)

Turns on or off the "auto commit" mode.

Returns **true** in case of success and **false** when the operation could not be performed or when it is not implemented.

9.5 Cursor objects

A cursor object contains methods to retrieve data resulting from an executed statement. A cursor object is created by using the connection:execute function.

9.5.1 cur:close()

Closes this cursor.

Returns **true** in case of success and **false** when the object is already closed.

9.5.2 cur:fetch([table[,modestring]])

Retrieves the next row of results.

If fetch is called without parameters, the results will be returned directly to the caller. If fetch is called with a table, the results will be copied into the table and the changed table will be returned. In this case, an optional modestring parameter can be used. It is just a string indicating how the resulting table should be constructed.

The mode string can contain:

- **n**: the resulting table will have numerical indices (default)
- **a**: the resulting table will have alphanumerical indices

The numerical indices are the positions of the fields in the SELECT statement; the alphanumerical indices are the names of the fields. The optional table parameter is a table that should be used to store the next row. This allows the use of a unique table for many fetches, which can improve the overall performance.

A call to fetch after the last row has already being returned will close the corresponding cursor. There is no guarantee about the types of the results: they may or may not be converted to adequate Lua types by the driver.

Returns **data**, as above, or **nil** if there are no more rows.

9.5.3 cur:getcolnames()

Returns a **list (table) of column names**.

9.5.4 cur:getcoltypes()

Returns a **list (table) of column types**.

9.6 SQLite3 extensions

Besides the basic functionality provided by all drivers, the SQLite3 driver also offers this extra feature:

9.6.1 env:connect(sourcename[,locktimeout])

In the SQLite3 driver, this method adds an optional parameter that indicate the amount of milliseconds to wait for a write lock if one cannot be obtained immediately. See also environment objects.

Returns a connection object.

9.6.2 conn:escape(str)

Escape especial characters in the given string according to the connection's character set. See also the official documentation of function sqlite3_mprintf.

Returns the **escaped string**.

CHAPTER 10

Sample code

Tip: If you want to see some ET-specific Lua examples, you can check the ET Legacy Lua scripts repository.

10.1 General

General example:

```
--[[
Lua Example ETLegacy
--]]

function et_InitGame(levelTime,randomSeed,restart)
    et.RegisterModname("Lua Example") -- Registering the modname.
    et.G_Print("Lua Loaded!\n") -- Printing out that the lua module has been loaded!
end

function et_ClientCommand(clientNum, command)
    local mapname = et.trap_Cvar_Get( "mapname" ) -- local varible storing the map␣
↪name
    if string.lower(command) == "mapname" then -- checking if the command is /mapname
        et.trap_SendServerCommand(clientNum, "chat \"Current Map:"..mapname.." \n\"")␣
↪-- printing out the map name to the client
    end
    return 1
end
```

This code does nothing useful besides demonstrate and exercise the Lua API:

```
-- printf wrapper
function et.G_Printf(...)
       et.G_Print(string.format(unpack(arg)))
```

(continues on next page)

(continued from previous page)

```
end


-- test some of the supported etpro lua functions
function test_lua_functions()
	et.trap_Cvar_Set( "bla1", "bla2" )
	et.G_Printf( "sv_hostname [%s]\n", et.trap_Cvar_Get( "sv_hostname" ) )
	et.G_Printf( "configstring 1 [%s] \n", et.trap_GetConfigstring( 1 ) )
	et.trap_SetConfigstring( 4, "yadda test" )
	et.G_Printf( "configstring 4 [%s]\n", et.trap_GetConfigstring( 4 ) )
	et.trap_SendConsoleCommand( et.EXEC_APPEND, "cvarlist *charge*\n" )
	et.trap_SendServerCommand( -1, "print \"Yadda yadda\"" )
	et.G_Printf( "gentity[1022].classname = [%s]", et.gentity_get( 1022, "classname
↪" ) )
end


-- called when game inits
function et_InitGame( levelTime, randomSeed, restart )
	et.G_Printf( "et_InitGame [%d] [%d] [%d]\n", levelTime, randomSeed, restart )
	et.RegisterModname( "bani qagame " .. et.FindSelf() )
--	test_lua_functions()
end


-- called every server frame
function et_RunFrame( levelTime )
	if math.mod( levelTime, 1000 ) == 0 then
--		et.G_Printf( et_RunFrame [%d]\n", levelTime )
	end
end


-- called for every clientcommand
-- return 1 if intercepted, 0 if passthrough
function et_ClientCommand( clientNum, cmd )
	et.G_Printf( "et_ClientCommand: [%d] [%s]\n", et.trap_Argc(), cmd )
	return 0
end


-- called for every consolecommand
-- return 1 if intercepted, 0 if passthrough
function et_ConsoleCommand()
	et.G_Printf( "et_ConsoleCommand: [%s] [%s]\n", et.trap_Argc(), et.trap_Argv(0)␣
↪)
	if et.trap_Argv(0) == "listmods" then
		i = 1
		repeat
			modname, signature = et.FindMod( i )
			if modname and signature then
				et.G_Printf( "vm slot [%d] name [%s] signature [%s]\n",
↪ i, modname, signature )
				et.IPCSend( i, "hello" )
			end
			i = i + 1
		until modname == nil or signature == nil
```

(continues on next page)

(continued from previous page)

```
                return 1
        end
        return 0
 end


-- called when we receive an IPC from another VM
function et_IPCReceive( vmnumber, message )
        et.G_Printf( "IPCReceive [%d] from [%d] message [%s]\n", et.FindSelf(),␣
↪vmnumber, message )
end


-- called for every ClientConnect
function et_ClientConnect( clientNum, firstTime, isBot )
        et.G_Printf( "et_ClientConnect: [%d] [%d] [%d]\n", clientNum, firstTime, isBot␣
↪)
--      return "go away"
        return nil
end


-- called for every ClientDisconnect
function et_ClientDisconnect( clientNum )
        et.G_Printf( "et_ClientDisconnect: [%d]\n", clientNum )
end


-- called for every ClientBegin
function et_ClientBegin( clientNum )
        et.G_Printf( "et_ClientBegin: [%d]\n", clientNum )
end


-- called for every ClientUserinfoChanged
function et_ClientUserinfoChanged( clientNum )
        et.G_Printf( "et_ClientUserinfoChanged: [%d] = [%s]\n", clientNum, et.trap_
↪GetUserinfo( clientNum ) )
end


-- called for every trap_Printf
function et_Print( text )
--      et.G_Printf( "et_Print [%s]", text )
end
```

10.2 Configstring

Example:

```
-- get the name of client #3 using configstrings
local cs = et.trap_GetConfigstring(et.CS_PLAYERS + 3)
local name = et.Info_ValueForKey(cs, "n")
```

10.3 Inter Process Communication (IPC)

Example scripts illustrating communication between these scripts using the et.IPCSend() and et_IPCReceive() functions.

10.3.1 Sender

Example of sender module:

```
--[[
ipcdemo-admin.lua
--]]

local IPCQueue = {}
local AdminGUIDs = {
        -- name,        guid,                                level
        { "Vetinari", "ABCDEF1234567890ABCDEF1234567890", 5 },
        { "Havelock", "1234567890ABCDEF1234567890ABCDEF", 3 }
    }

function et_InitGame(levelTime, randomSeed, restart)
     et.RegisterModname("ipcdemo-admin.lua")
end

function et_IPCReceive(vm, msg)
    local level
    local junk1, junk2, id = string.find(msg, "IsAdmin:%s+(%d+)")
    if id ~= nil then
        id    = tonumber(id)
        guid  = et.Info_ValueForKey(et.trap_GetUserinfo(id), "cl_guid")
        level = table.foreach(AdminGUIDs,
            function(i, admin)
                if admin[2] == guid then
                    return(admin[3])
                end
            end
        )
        if level == nil then
            level = 0
        end
        table.insert(IPCQueue, { vm, level, id })
    end
end

function et_RunFrame(lvltime)
    table.foreach(IPCQueue,
        function(i, queue)
            local ok = et.IPCSend(queue[1],
                           string.format("IsAdmin: %d %d", queue[2], queue[3]))
            if ok ~= 1 then
                local mod, cksum = et.FindMod(queue[1])
                et.G_Print(string.format("ipcdemo-admin: IPCSend to %s (vm: %d) failed
↪", mod, queue[1]))
            end
        end
    )
```

(continues on next page)

(continued from previous page)

```
    IPCQueue = {}
end
```

10.3.2 Receiver

Example of receiver module:

```
--[[
ipcdemo-cmd.lua
--]]

local admin_vm    = -1
local CommandQueue = {}

function et_InitGame(levelTime, randomSeed, restart)
    local mod = ""
    local sig = ""
    local i = 1
    while mod ~= nil do
        mod, sig = et.FindMod(i)
        if string.find(mod, "^ipcdemo-admin.lua") == 1 then
            admin_vm = i
            mod      = nil
        end
        i = i + 1
    end
    if admin_vm == -1 then
        et.G_Print("ipcdemo-cmd.lua: Could not find vm number for ipcdemo-admin.lua")
    end
    et.RegisterModname("ipcdemo-cmd.lua")
end

function et_IPCReceive(vm, msg)
    if vm == admin_vm then
        local junk1,junk2,level,id = string.find(msg, "IsAdmin:%s+(%d+)%s+(%d)")
        if level ~= nil and id ~= nil then
            runAction(tonumber(id), tonumber(level))
        end
    end
end

function runAction(id, level)
    local done = table.foreach(CommandQueue,
        function(i, queue)
            if id == queue[1] then
                if queue[2] <= level then
                    if queue[4] == nil then
                        et.trap_SendConsoleCommand(et.EXEC_INSERT, queue[3])
                    else
                        et.trap_SendConsoleCommand(et.EXEC_INSERT,
                                string.format("%s %s", queue[3], queue[4]))
                    end
                end
                return(i)
            end
```

(continues on next page)

(continued from previous page)

```
        end
    )
    if done ~= nil then
        table.remove(CommandQueue, done)
    end
end

function et_ClientCommand(id, command)
    local arg0 = et.trap_Argv(0)
    local arg1 = et.trap_Argv(1)
    if arg0 == "say" then
        if arg1 == "!axis" then
            --          id, lvl, cmd,          argument
            queueCommand(id, 4, "forceteam r", id)
        elseif arg1 == "!allies" then
            queueCommand(id, 4, "forceteam b", id)
        elseif arg1 == "!shuffle" then
            queueCommand(id, 3, "shuffleteamsxp_norestart", nil)
        end
    end
    return(0)
end

function queueCommand(id, level, cmd, argument)
    if admin_vm ~= -1 then
        local ok = et.IPCSend(admin_vm, string.format("IsAdmin: %d", id))
        if ok ~= 1 then
            et.G_Print("ipcdemo-cmd: IPCSend to ipcdemo-admin failed")
        else
            table.insert(CommandQueue, { id, level, cmd, argument })
        end
    end
end
```

10.4 Database

Exemple using LuaSQL.

10.4.1 Basic use

Here is an example of the basic use of the library:

```
--[[
LuaSQL demo
--]]

-- load driver
local driver = require "luasql.sqlite3"
-- create environment object
env = assert (driver.sqlite3())
-- connect to data source
con = assert (env:connect("luasql-test"))
-- reset our table
```

(continues on next page)

(continued from previous page)

```
res = con:execute"DROP TABLE people"
res = assert (con:execute[[
  CREATE TABLE people(
    name  varchar(50),
    email varchar(50)
  )
]])
-- add a few elements
list = {
  { name="Jose das Couves", email="jose@couves.com", },
  { name="Manoel Joaquim", email="manoel.joaquim@cafundo.com", },
  { name="Maria das Dores", email="maria@dores.com", },
}
for i, p in pairs (list) do
  res = assert (con:execute(string.format([[
    INSERT INTO people
    VALUES ('%s', '%s')]], p.name, p.email)
  ))
end
-- retrieve a cursor
cur = assert (con:execute"SELECT name, email from people")
-- print all rows, the rows will be indexed by field names
row = cur:fetch ({}, "a")
while row do
  et.G_Print("Name:" .. row.name .. ", E-mail: " .. row.email .."\n")
  -- reusing the table of results
  row = cur:fetch (row, "a")
end
-- close everything
cur:close() -- already closed because all the result set was consumed
con:close()
env:close()
```

And the output of this script should be:

```
Name: Jose das Couves, E-mail: jose@couves.com
Name: Manoel Joaquim, E-mail: manoel.joaquim@cafundo.com
Name: Maria das Dores, E-mail: maria@dores.com
```

10.4.2 Iterator

Here is how to create an iterator over the result of a SELECT query:

```
function rows (connection, sql_statement)
  local cursor = assert (connection:execute (sql_statement))
  return function ()
    return cursor:fetch()
  end
end
```

Here is how the iterator is used:

```
env = assert (require"luasql.mysql".mysql())
con = assert (env:connect"my_db")
for id, name, address in rows (con, "select * from contacts") do
```

(continues on next page)

(continued from previous page)

```
  print (string.format ("%s: %s", name, address))
end
```

Obviously, the code above only works if there is a table called contacts with the columns id, name and address in this order. At the end of the loop the cursor will be automatically closed by the driver.