CScriptObjectGame Reference

2004-09-13

Table of Contents

CScriptObjectGame: C++ functions available in Lua script	
`GetCDPath ()`	**Usage:** Gets the path to the cdrom drive. Used to play e.g. cutscenes from the Far Cry cd. **Parameters:** none **Return:** Returns a string, nil if failed. **Code Example:** local szCDPath = Game:GetCDPath();
`GetUserName ()`	**Usage:** Gets the user name / player name. **Parameters:** none **Return:** Returns a string. **Code Example:** setglobal("sv_name", Game:GetUserName().."'s Server");
`Load (string)`	**Usage:** Loads the game from a file. Takes the name of the target file [optional]. The default is "farcry_save.sav" **Parameters:** none **Return:** none **Code Example:** not used

CScriptObjectGame: C++ functions available in Lua script	
`GetPlayers ()`	**Usage:** Gets all player entities in game. **Parameters:** none **Return:** // _SmartScriptObject pObj(m_pScriptSystem); *pObj, table filled with all player entities in game. **Code Example:** local PlayerList = Game:GetPlayers();
`SetHUDFont (string, string)`	**Usage:** Set the font used by the functions WriteHudStrings and WriteHudNumber. **Parameters:** string: Fontname string enumerating the font name. string: Effectname string enumerating the font shader. **Return:** none **Code Example:** Game:SetHUDFont("radiosta", "binozoom");

CScriptObjectGame: C++ functions available in Lua script	
`WriteHudNumber (int, int, int, float, float, float, float, float, float)`	**Usage:** Print a string into the Hud. **Parameters:** int: X coordinate into the screen (the screen is always normalized to 800x600). int: Y coordinate into the screen (the screen is always normalized to 800x600). int: Number to print. float: Red component of the color used to print the number. float: Green component of the color used to print the number. float: Blue component of the color used to print the number. float: Witdh of a single character. float: Height of a single character. **Return:** none **Code Example:** not used

CScriptObjectGame: C++ functions available in Lua script

```
WriteHudString

(int,
 int,
 string,
 float,
 float,
 float,
 float,
 float,
 float,
 bool)
```

Usage:
Print a string into the Hud with variable size fonts (a letter 'm' is wider than 'i')

Parameters:
int: X coordinate into the screen (the screen is always normalized to 800x600).

int: Y coordinate into the screen (the screen is always normalized to 800x600).

string: String string to print.

---

float: Red component of the color used to print the number.

float: Green component of the color used to print the number.

float: Blue component of the color used to print the number.

float: Alpha component of the color used to print the number.

---

float: Width of a single character.

float: height of a single character.

bool: Center the message on screen

Return:
Returns the starting pos if center was true.

Code Example:
Game:WriteHudString(10, 100, "@"..Hud.PlayerObjective, 1, 1, 1, 1, 30, 30);

CScriptObjectGame: C++ functions available in Lua script

```
WriteHudStringFixed

(int,
 int,
 string,
 float,
 float,
 float,
 float,
 float,
 float,
 float)
```

Usage:
Print a string into the Hud with fixed size (both letter 'm' and 'i' have the same width).

Parameters:
int: X coordinate into the screen (the screen is always normalized to 800x600).

int: Y coordinate into the screen (the screen is always normalized to 800x600).

string: String string to print.

---

float: Red component of the color used to print the number.

float: Green component of the color used to print the number.

float: Blue component of the color used to print the number.

float: Alpha component of the color used to print the number.

---

float: Width of a single character.

float: height of a single character.

float: A width-scale ratio.

Return: none

Code Example:
Game:WriteHudStringFixed(posZoomX, posZoomY, s, 1, 0, 0, 1 , 10, 10, 1.0);

CScriptObjectGame: C++ functions available in Lua script	
`GetHudStringSize (string, float, float, float)`	**Usage:** Gets the size x and y sizes of a passed string with a certain letter size. **Parameters:** string: String to get the size from. float: X size of the text (10.0f by default). float: Y size of the text (10.0f by default). float: [Optional] WrapWidth, if bigger then 0, then function returns the text sizes, according to this value, with fixed size (both letter 'm' and 'i' have the same width). **Return:** Returns two variables: stringXSize, stringYSize, **Code Example:** local fieldSpaceSize = %Game:GetHudStringSize(" ", header_textsize, header_textsize);
`GetServerList ()`	**Usage:** Gets the list of servers on the network with related information. **Parameters:** none **Return:** // _SmartScriptObject pObj(m_pScriptSystem); *pObj, a table with the server infos. **Code Example:** local ServerList = Game:GetServerList();
`GetMaterialIDByName (string)`	**Usage:** Gets the corresponding material id to a passed material name. **Parameters:** string: The name of the material, we need the id from. **Return:** Returns the material id or nil if the material does not exist or if it is not loaded in the current map. **Code Example:** hit.target_material = Game:GetMaterialBySurfaceI (Game:GetMaterialIDByName("mat_head"));

CScriptObjectGame: C++ functions available in Lua script	
`ReloadMaterialPhysics (string)`	**Usage:** Reloads the material propperties of all surfaces with the passed materialname. **Parameters:** string: Name of the material to update. **Return:** none **Code Example:** not used
`GetActions ()`	**Usage:** Gets a list of possible actions, depends on the current action map. **Parameters:** none **Return:** // _SmartScriptObject pObj(m_pScriptSystem); *pObj, a table with the current actions. **Code Example:** local ActionList = Game:GetActions();
`IsPlayer (int)`	**Usage:** Check if an entity is the player or not. **Parameters:** int: Represents the entity id. **Return:** !=nil: passed entity is the player nil: passed entity is not the player **Code Example:** not used
`GetEntitiesScreenSpace (string)`	**Usage:** Gets a list of entities, which are visible. Optionally use a bone as center instead of the bounding box center. **Parameters:** string: [Optional] A bone name. **Return:** // _SmartScriptObject pTable(m_pScriptSystem); *pTable, containing a list of visible entities. **Code Example:** local pEntities = Game:GetEntitiesScreenSpace("Bip01 Head");

CScriptObjectGame: C++ functions available in Lua script	
`GetPlayerEntitiesInRadius` `(vector3, float, table, int)`	**Usage:** Get a list of player entities within a certain radius. **Parameters:** vector3: Center of the radius float: The radius, within the function checks. Table: Store the found entities in here. int: [Optional] 0 = returns alive and trackable entities only, 1 = returns all entities **Return:** none **Code Example:** Game:GetPlayerEntitiesInRadius(pos, radius, players);

CScriptObjectGame: C++ functions available in Lua script

```
DrawRadar

(float,
 float,
 float,
 float,
 float,
 int,
 int,
 int,
 int,
 int,
 int,
 int,
 table,
 string)
```

Usage:
Draw the radar with certain textures, position, etc... on screen.

Parameters:
float: X position of the radar.

float: Y position of the radar.

float: Width of the radar.

float: Height of the radar.

float: The range of the radar.

// Textures in dds format:
int: ID of radar texture 1.

int: ID of radar texture 2.

int: ID of radar texture 3.

int: ID of radar texture 4.

int: ID of radar texture 5.

int: ID of radar texture 6.

int: ID of radar texture 7.

---

table: A table of entities to show up.

string: The radar objective(s).

Return: none

Code Example:
Game:DrawRadar(x, y, w, h,
tonumber(g_RadarRange),
self.Radar,
self.RadarMask,
self.RadarPlayerIcon,
self.RadarEnemyInRangeIcon, self.RadarEnemyOutRangeIcon,
self.RadarSoundIcon,
self.RadarObjectiveIcon,
Hud.tblPlayers,
RadarPosition);

CScriptObjectGame: C++ functions available in Lua script

```
DrawHalfCircleGauge

(float,
 float,
 float,
 float,
 float,
 float,
 float,
 float,
 int,
 float,
 float,
 float,
 float,
 float)
```

Usage:
Render a half circle gauge (status bar like?).

Parameters:

// Sizes of the gauge
float: X

float: Y

float: Width

float: Height

// Texture coordinates
float: U

float: V

float: UW

float: VH

int: An id to the texture we use.

float: The value of the status (0 - 100)

// Color values
float: Red

float: Green

float: Blue

float: Alpha

Return:
Returns the first parameter.

Code Example: not used

CScriptObjectGame: C++ functions available in Lua script	

`ShowIngameDialog`
`(int, string, string, int, string, float)`
	Usage:
	Shows a dialog on the screen with a given string.
	
	Parameters:
	int: The fill id, is always used with -1 in scripts.
	
	string: The name of the font to use.
	
	string: The name of the shadereffect to use.
	
	int: The size of the dialog.
	
	string: The message itself.
	
	float: The timeout setting. (for fading?)
	
	Return:
	Returns an integer nId, probably the id of the created dialog. Check CIngameDialogMgr::AddDialog() for details.
	
	Code Example:
	Game:ShowIngameDialog(-1, "", "", 12, "You need the "..KeyCardInfo[self.Properties.nNeededKey].Desc.." to open this door...", 3);

`HideIngameDialog`
`(int)`
	Usage:
	Hides an ingame dialog again.
	
	Parameters:
	int: The number or id of the dialog to hide.
	
	Return: none
	
	Code Example:
	Game:HideIngameDialog(self.DialogId);

CScriptObjectGame: C++ functions available in Lua script	

`EnableUIOverlay` `(int, int)`	**Usage:**
	Shows or hides the user interface overlay.
	
	Parameters:
	int: 1 = Enable the overlay, draw it then.
	0 = Default value, disable the overlay.
	
	int: 1 = Set exclusive input rights.
	0 = Default value, non exclusive input.
	
	Return: none
	
	Code Example:
	// Disable user interface overlay
	Game:EnableUIOverlay(0, 0);
	
	// Enable user interface overlay
	Game:EnableUIOverlay(1, 1);

`IsUIOverlay` `()`	**Usage:** Checks if we have the user interface enabled or not. **Parameters:** none **Return:** != nil if overlay is true. nil if false. **Code Example:** not used
`GetEntityTeam` `(int)`	**Usage:** Passes an entity id and gets the name of the team it belongs to. **Parameters:** int: The entity id. **Return:** Returns the team name or nil, if the entity does not belong to a team. **Code Example:** local targetTeam=Game:GetEntityTeam(target.id);

CScriptObjectGame: C++ functions available in Lua script	
`GetTeamScore (string)`	**Usage:** Gets the score of a certain team. **Parameters:** string: The team name, we need to know the score from. **Return:** Returns an integer, the team score. Returns nil if the team does not exist. **Code Example:** local red_score=Game:GetTeamScore("red");
`GetTeamFlags (string)`	Seems to be the same as the one above, but does not send an assert if the teamname was nil.
`Connect (string, bool, bool)`	**Usage:** Creates a local client and conects it to the server. **Parameters:** string: Server string, containing the server name or the ip number. bool: DoLateSwitch, true or false. (1 or nil) This is set to false by default. bool: DoCDAuthorization, do we need a cd key? This is set to false by default. **Return:** none **Code Example:** Game:Connect(UI.PageLANServerList.szJoinIP, 1);
`Reconnect ()`	**Usage:** Creates a local client and connects to the last server. **Parameters:** none **Return:** none **Code Example:** Game:Reconnect();

CScriptObjectGame: C++ functions available in Lua script	
`Disconnect` `()`	**Usage:** Disconnects the current connection to a remote server. **Parameters:** none **Return:** none **Code Example:** Game:Disconnect();
`GetLevelList` `(string)`	**Usage:** Lists all levels which belong to a certain mission. **Parameters:** string: The name of a mission. **Return:** Returns the list of levels which belong to the passed mission. If no mission name is passed, all levels will be returned. **Code Example:** local LevelList = Game:GetLevelList();
`LoadLevel` `(string, string)`	**Usage:** Loads a level, starts a local client and connects it to the local server, no external connections (sp game). **Parameters:** string: This is the name of the map to load. string: [Optional] This is the name of the mission. Otherwise an empty string will be used. **Return:** none **Code Example:** not used
`GetLevelName` `()`	**Usage:** Gets the name of the current level. **Parameters:** none **Return:** Returns the level name.

CScriptObjectGame: C++ functions available in Lua script	
`LoadLevelListen (string, string)`	**Usage:** Loads a level, starts a local client and connects it to the local server. Allows external connections (mp). **Parameters:** string: This is the name of the map to load. string: [Optional] This is the name of the mission. Otherwise an empty string will be used. **Return:** none **Code Example:** Game:LoadLevelListen(getglobal('gr_NextMap'));
`LoadLevelMPServer (string, string)`	**Usage:** Loads a level on a mp server, keeping the current clients connected to the current server. **Parameters:** string: This is the name of the map to load. string: [Optional] This is the name of the mission. Otherwise an empty string will be used. **Return:** none **Code Example:** Game:LoadLevelMPServer(getglobal('gr_NextMap'));
`GetVersion (string)`	**Usage:** Get the game version as a string. **Parameters:** string: [Optional] Formation string used for this function (second param) in C++: sprintf(string, string, bool, bool, word), **Return:** Returns the game version as a string. **Code Example:** text = "v"..Game:GetVersion("%d.%d © 2004 Crytek, All Rights Reserved");

CScriptObjectGame: C++ functions available in Lua script	
`GetVersionString ()`	**Usage:** Gets the version of the game as a string. **Parameters:** none **Return:** Returns the game version as a string. **Code Example:** not used
`CreateVariable (string, value, string)`	**Usage:** Creates a console variable. **Parameters:** string: The name of the console variable. value: [Optional] A default value,, string or number. string: [Optional] A user definded flag, which is used by other subsystems and does not affect the console variable (basically of user data). **Return:** none **Code Example:** Game:CreateVariable("hud_damageindicator",1);
`RemoveVariable (string)`	**Usage:** Removes a console variable. **Parameters:** string: The name of the variable to remove. **Return:** none **Code Example:** not used
`SetVariable (string, value)`	**Usage:** Sets a value to a variable. **Parameters:** string: The name of the variable to set the value to. value: The value itself, string or number. **Return:** None on success, nil when failed. **Code Example:** Game:SetVariable(szVarName, 0);

CScriptObjectGame: C++ functions available in Lua script	
`GetVariable (string)`	**Usage:** Gets the value of a variable. **Parameters:** string: The name of the variable. **Return:** Nil if failed, otherwise the value, a string or number (int or float). **Code Example:** local szValue = Game:GetVariable(szVarName);
`Save (string)`	**Usage:** Saves the game in a file. **Parameters:** string: [Optional] Name of the target file. Default name is farcry_save.sav **Return:** none **Code Example:** not used
`Quit ()`	**Usage:** Quits the game. **Parameters:** none **Return:** none **Code Example:** not used
`IsPointInWater (vector3)`	**Usage:** Checks if a specifique point is under water level or not. **Parameters:** vector3: A table, containing x, y, z positions of the point to test. **Return:** Returns != nil if true nil if false **Code Example:** if (Game:IsPointInWater(Params.pos) == nil) then ...

CScriptObjectGame: C++ functions available in Lua script	
`GetWaterHeight (vector3)`	**Usage:** Get the water height level. **Parameters:** vector3: [Optional] A table, containing x, y, z positions of the point where we want to get the water height. **Return:** If no point is passed, the function returns the water height (z value) by using the player (visibile area) position. Otherwise, it returns the water height level (z value) at the given point. **Code Example:** vVec.z = Game:GetWaterHeight() + 0.02;
`RefreshServerList ()`	**Usage:** Refreshes the server list from the LAN network. **Parameters:** none **Return:** none **Code Example:** Game:RefreshServerList();
`ClearServerInfo ()`	**Usage:** Clears the m_hmServerTable in CNETServerSnooper. **Parameters:** none **Return:** none **Code Example:** not used
`GetServerInfo (string int)`	**Usage:** Gets necessary server info for creating a game server from a client. Adds the ip to the master server (list). **Parameters:** string: The server ip or name. int: The server port. **Return:** Returns nil if one of the parameters failed to load. Otherwise it returns 1; **Code Example:** Game:GetServerInfo(szIP, szPort);

CScriptObjectGame: C++ functions available in Lua script	
`GetServerListInfo (CScriptObjectVector)`	**Usage:** Adds a list of servers and their information to the master server. **Parameters:** CScriptObjectVector: A list of server ips. **Return:** Returns nil if failed and 1 on success. **Code Example:** none
`ExecuteRConCommand (string)`	**Usage:** Executes a remote control system command. **Parameters:** string: The name of the command to execute over the RCS (remote control system). **Return:** Returns nil if failed and 1 on success. **Code Example:** not used
`IsServer ()`	**Usage:** Checks is the local host is a server or not. **Parameters:** none **Return:** != nil (true) if the local host is a server. nil if the local host is no server. **Code Example:** if(Game:IsServer())then ...
`IsClient ()`	**Usage:** Checks is the local host is a client or not. **Parameters:** none **Return:** != nil (true) if the local host is a client. nil if the local host is no client. **Code Example:** if (Game:IsClient()) then ...

CScriptObjectGame: C++ functions available in Lua script	
`IsMultiplayer ()`	**Usage:** Check if we are in multiplayer mode or not. **Parameters:** none **Return:** != nil (true) if we are in multiplayer mode, (being either a server or a client). nil if we are not in multiplayer mode. **Code Example:** if (not Game:IsMultiplayer()) then ...
`SetTimer (table, float, table)`	**Usage:** This function sets a timer callback. **Parameters:** table: The table object that will receive the OnEvent with ScriptEvent_Timer as eventid. float: Duration on the timer in milliseconds. table: [Optional] Table that will be passed back by the callback. **Return:** Returns the id timer. **Code Example:** Game:SetTimer(MuzzleFlashTurnoffCallbackVC, lifetime, MuzzleFlashParams);
`KillTimer (int)`	**Usage:** Snoozes a timer event. **Parameters:** int: The timer id returned by Game:SetTimer(). **Return:** none **Code Example:** Game:KillTimer(self.Timer);

CScriptObjectGame: C++ functions available in Lua script	
`StartRecord (string)`	**Usage:** Records a demo and saves it. **Parameters:** string: [Optional] The name of the demo. **Return:** none **Code Example:** not used
`StopRecord ()`	**Usage:** Stops recording a demo. **Parameters:** none **Return:** none **Code Example:** not used
`StartDemoPlay (string)`	**Usage:** Plays back a recorded demo. **Parameters:** string: The name of the demo to play. **Return:** none **Code Example:** Game:StartDemoPlay(name);
`StopDemoPlay ()`	**Usage:** Stops playing a demo. **Parameters:** none **Return:** none **Code Example:** not used
`DisplayNetworkStats ()`	**Usage:** UNDER DEVELOPMENT. NOT USED CURRENTLY AND DOES NOT DO ANYTHING! **Parameters:** none **Return:** none **Code Example:** not used

CScriptObjectGame: C++ functions available in Lua script	
`ForceScoreBoard` `(int, bool)`	**Usage:** Removes the scoreboard to a certain client connected to this server. **Parameters:** int: The id of the player entity associated if this parameter is 0 brodcast the command to all clients. bool: If != nil activate the scoreboard, if nil decativate it. **Return:** Returns, the current team score or nil if the specified team doesn't exist. **Code Example:** Game:ForceScoreBoard(Slot:GetPlayerId(), yes);
`ReloadMaterials` `()`	**Usage:** Reloads all material scripts. **Parameters:** none **Return:** none **Code Example:** not used
`GetTagPoint` `(string)`	**Usage:** Gets the position of a certain tagpoint. **Parameters:** string: The name of the searched tagpoint. **Return:** Returns a vector3 (table) with the positions of the passed tagpoint. **Code Example:** local TagPoint = Game:GetTagPoint(run_target);

CScriptObjectGame: C++ functions available in Lua script	
`GetMaterialBySurfaceID (int)`	**Usage:** Gets the material table to a passed material id. **Parameters:** int: The id of the material. **Return:** Returns the material table or nil if the specified id is not related to any loaded material. **Code Example:** hit.target_material = Game:GetMaterialBySurfaceID(Game:GetMaterialIDByName("mat_head"));
`ReloadWeaponScripts ()`	**Usage:** Reloads all weapon scripts. **Parameters:** none **Return:** none **Code Example:** not used
`AddWeapon (string)`	**Usage:** Adds a weapon to the weapon system. **Parameters:** string: The name of the weapon to add. **Return:** Returns an error string, if weapon was not loaded and so not valid. Otherwise, returns none. **Code Example:** Game:AddWeapon(wName);
`GetWeaponClassIDByName (string)`	**Usage:** Gets the id of a certain weapon. **Parameters:** string: The string of the weapon we need the id of. **Return:** Returns the weapon id as a number, otherwise on fail, the function returns nil. **Code Example:** local weaponid = Game:GetWeaponClassIDByName(item.Name);

CScriptObjectGame: C++ functions available in Lua script	
`SetThirdPerson (bool)`	**Usage:** Sets the camera to third person mode and back. **Parameters:** bool: != nil means true, nil means false **Return:** none **Code Example:** Game:SetThirdPerson(0); // first person mode
`SetViewAngles (vector3)`	**Usage:** Sets the view angles of the camera. **Parameters:** vector3: A table, containing the angles. **Return:** none **Code Example:** not used
`DumpEntities ()`	**Usage:** Dumps all existing entities. Continues the deleting loop when a projectile was found. **Parameters:** none **Return:** none **Code Example:** not used
`TouchCheckPoint (int, table, table)`	**Usage:** Makes a call with the current checkpoint number and saves the game for this checkpoint. **Parameters:** int: The id of this checkpoint. table: The position of the checkpoint. table: The angles of the checkpoint. **Return:** none **Code Example:** Game:TouchCheckPoint(self.Properties.nId, _LastCheckPPos, _LastCheckPAngles);

CScriptObjectGame: C++ functions available in Lua script	
`LoadLatestCheckPoint ()`	**Usage:** Loads the game at the latest saved check point status. **Parameters:** none **Return:** none **Code Example:** not used
`ShowSaveGameMenu ()`	**Usage:** Checks if the save game menu is shown up. **Parameters:** none **Return:** != nil means true nil means false **Code Example:** if(Game:ShowSaveGameMenu()) then ...
`GetSaveGameList (string)`	**Usage:** Gets a list of all save-games. **Parameters:** string: The profile name of the player. **Return:** Returns a list (table) with all corresponding saved games. **Code Example:** local SaveList = Game:GetSaveGameList(getglobal("g_playerprofile"));
`ToggleMenu ()`	**Usage:** Toggles the menu on and off by sending a switch message (popup effect). **Parameters:** none **Return:** none **Code Example:** not used

CScriptObjectGame: C++ functions available in Lua script	

`ShowMenu`
`()`
	Usage:
	Switches the game to the menu.
	
	Parameters: none
	
	Return: none
	
	Code Example:
	Game:ShowMenu();

`HideMenu`
`()`
	Usage:
	Switches to the game again.
	
	Parameters: none
	
	Return: none
	
	Code Example:
	Game:HideMenu();

`IsInMenu`
`()`
	Usage:
	Checks if the game is in a menu or not.
	
	Parameters: none
	
	Return:
	Returns 1 if true,
	nil if we are non in a menu.
	
	Code Example:
	if (not Game:IsInMenu()) then
	...

`SendMessage`
`(string)`
	Usage:
	Sends a message to the game (appears as message in game?)
	
	Parameters:
	string: The message to send.
	
	Return:
	Returns an error string if passed parameter is nil, otherwise the function returns none.
	
	Code Example:
	Game:SendMessage("LoadGame "..szFilename);

CScriptObjectGame: C++ functions available in Lua script	
`GetEntityClassIDByClassName (string)`	**Usage:** Gets the entity class id by using its name. **Parameters:** string: The name of the entity. **Return:** Returns an integer number, the class id. **Code Example:** local classid=Game:GetEntityClassIDByClassName("FlagEntity");
`SetCameraFov (float)`	**Usage:** Sets the camera field of view to the passed value. **Parameters:** float: The angle for the fov. Default is ½ PI. **Return:** none **Code Example:** Game:SetCameraFov(self.NoZoom);
`GetCameraFov ()`	**Usage:** Gets the current camera fov value. **Parameters:** none **Return:** Returns the camera fov. **Code Example:** local shift = xcent * tan(0.1308997)/tan(Game:GetCameraFov()/2.0) * factor;
`ApplyStormToEnvironment (vector3, float)`	**Usage:** This function applies a storm effect, meaning wind and rain (if outddors) to the player position (visibility area). This effect is client sided only! **Parameters:** vector3: The wind direction as a vector (table). float: The amount of rain to show. **Return:** Returns always nil. **Code Example:** Game:ApplyStormToEnvironment(self.Properties.vWindDir, self.fCurrentRain);

CScriptObjectGame: C++ functions available in Lua script	

`CreateExplosion` `(table)`	**Usage:**
	Creates an explosion.
	
	Parameters:
	table: A table, containing a lot of information about the explosion. Here is an example from the Grenade.lua file:
	
	ExplosionParams =
	{
	    pos = {},
	        damage = 150,
	        rmin = 0.8,
	        rmax = 8.5, -- default = 10.5
	        radius = 8.5, -- default = 8
	        DeafnessRadius = 10.5,
	        DeafnessTime = 12.0,
	        impulsive_pressure = 15, -- default 5
	        shooter = nil,
	        weapon = nil,
	        explosion = 1,
	        rmin_occlusion = 0.2,
	        occlusion_res = 32,
	        inflate = 2,
	}
	
	Return: none
	
	Code Example:
	Game:CreateExplosion(self.ExplosionParams);

`DrawLabel` `(vector3, float, string)`	**Usage:** Draws a text label. Only used in the Waypoint.lua file. **Parameters:** vector3: A position where to draw the label. float: The label size as a single number. **Return:** none **Code Example:** Game:DrawLabel(pos, self.Properties.LabelSize, Language[self.Properties.LabelText]);

CScriptObjectGame: C++ functions available in Lua script

`GetInstantHit (table)`	**Usage:**
	Gets information about an object we 'hit'. (Seems to work like a trace)
	
	Parameters:
	table: A table, containing information about the hitting entity, like the player. It should contain:
	shooter,
	id,
	pos,
	dir,
	distance,
	
	Return:
	Returns a table with the following elements:
	// entity = 0
	// stat obj = 1
	// terrain = 3
	target,
	
	shooter,
	objtype,
	pos,
	normal,
	dir,
	target_material,
	
	Code Example: not used

CScriptObjectGame: C++ functions available in Lua script	

`GetMeleeHit (table)`	**Usage:**
	Gets information about a close object we 'hit'. (Seems to work like a trace)
	
	Parameters:
	table: A table, containing information about the hitting entity, like the player. It should contain:
	shooter,
	id,
	pos,
	dir,
	distance,
	melee_target,
	Return:
	Returns a table with the following elements:
	// entity = 0
	// stat obj = 1
	// terrain = 3
	target,
	
	shooter,
	objtype,
	pos,
	normal,
	dir,
	target_material,
	
	Returns nil if failed (no close object in bbox).
	
	Code Example: not used

`SaveConfiguration (string)`	**Usage:** Saves a profile configuration. **Parameters:** string: [Optional] The profilename of the player. Will be added to the path during saving: profiles/player/profilename **Return:** none **Code Example:** Game:SaveConfiguration(g_playerprofile);
`LoadConfiguration (string)`	**Usage:** Loads the system and game configuration. **Parameters:** string: [Optional] A profile name. If none is passed, then don't use profiles. **Return:** none

CScriptObjectGame: C++ functions available in Lua script	
`LoadConfigurationEx (string, string)`	**Usage:** Loads a system or game configuration, or both. **Parameters:** string: This is the name of the system configuration. string: This is the name of the game configuraton. **Return:** none **Code Example:** Game:LoadConfigurationEx("", szFileName);
`RemoveConfiguration (string)`	**Usage:** Removes the existing game and system configurations, needs a profile passed as single parameter. **Parameters:** string: A profile name. **Return:** none **Code Example:** Game:RemoveConfiguration(ProfileName);
`DrawHealthBar ()`	**Usage:** THIS FUNCTION IS EMPTY AND RETURNS IMMEDIATELY! **Parameters:** none **Return:** none **Code Example:** not used
`__RespawnEntity (int)`	**Usage:** Removes and the respawns a specified entity. **Parameters:** int: This is the id of the entity to respawn. **Return:** none **Code Example:** not used

CScriptObjectGame: C++ functions available in Lua script	

`ListPlayers`
`()`
	Usage:
	Prints a list of current players to the console.
	
	Parameters: none
	
	Return: none
	
	Code Example: not used

`LoadScript`
`(string,`
` bool)`
	Usage:
	Loads a script. Forces a reload if specified.
	
	Parameters:
	string: This is the exact path to the script.
	
	bool: [Optional] Should the script be reloaded if it already is loaded? Set to false by default.
	
	Return: none
	
	Code Example: not used

`ForceEntitiesToSleep`
`()`
	Usage:
	Iterates through the list of entities and sets them to sleep. Also works for ai entities, except the player!
	
	Parameters: none
	
	Return: none
	
	Code Example: not used

`CreateRenderer`
`()`
	Usage:
	Creates a new renderer on the renderer stack.
	
	Parameters: none
	
	Return:
	Returns a new CScriptObjectRenderer object.
	
	Code Example:
	self.rend = Game:CreateRenderer();

CScriptObjectGame: C++ functions available in Lua script	

`SoundEvent`
`(CScriptObjectVector,`
` float,`
` float,`
` int)`
	Usage:
	Generates a sound event on the radar.
	
	Parameters:
	CScriptObjectVector: The position of the sound.
	
	float: The radius.
	
	float: the intensity of the threat.
	
	int: The sound id, will be typecasted to an entityid.
	
	Return: none
	
	Code Example:
	Game:SoundEvent(pos,sound.SoundRadius, sound.Threat, self.ExplosionParams.shooterid);

`CheckMap`
`(string,`
` string)`
	Usage:
	Used to check if a map is ok or not, meaning that all the related stuff can be loaded.
	
	Parameters:
	string: The map name, do not include the path!
	
	string: [Optional] The game type. So this function will also check the xml file.
	
	Return:
	Returns 1 if map is ok,
	nil if it could not be loaded propperly.
	
	Code Example:
	if (not Game:CheckMap(mapname, szGameType)) then
	if (Game:CheckMap(mapname)) then
	...

`GetMapDefaultMission`
`(string)`
	Usage:
	Gets the default mission type for a specified map.
	
	Parameters:
	string: The map name.
	
	Return:
	Returns the name of the default mission for the passed map.
	
	Code Example: not used

CScriptObjectGame: C++ functions available in Lua script	
`CleanUpLevel ()`	**Usage:** This function cleans up the current level and makes it ready for quitting the game. It is called on terminate game. **Parameters:** none **Return:** none **Code Example:** Game:CleanUpLevel();
`SavePlayerPos (string, string)`	**Usage:** Saves the player position with a passed name and a description. **Parameters:** string: A name for the position to save. string: A description of the position. **Return:** none **Code Example:** not used
`LoadPlayerPos (string)`	**Usage:** Loads a previously saved player position. **Parameters:** string: The name of the position to load. **Return:** none **Code Example:** not used
`PlaySubtitle (USER_DATA)`	**Usage:** Plays a subtitle sound. **Parameters:** USER_DATA: This is a sound id, returned by LoadSound(IFunctionHandler * pH), a method of CScriptObjectSound. This could be an integer, an id so to speak. **Return:** none **Code Example:** Game:PlaySubtitle(self.sound);

CScriptObjectGame: C++ functions available in Lua script	
`GetModsList ()`	**Usage:** Get a table with the current mods, including the title, name, author, etc... **Parameters:** none **Return:** Returns a _SmartScriptObject, a table in lua. **Code Example:** not used
`LoadMOD (string, bool)`	**Usage:** Sets a mod to current mod. Restarts it if specified. **Parameters:** string: The name of the mod to set. bool: [Optional] If set to true (1= nil), the mod will do a restart. Set to false, by default. **Return:** Writes a success or failed message to the logfile. **Code Example:** Game:LoadMOD(tMod.Name,1);
`GetCurrentModName ()`	**Usage:** Gets the current mod name. **Parameters:** none **Return:** Returns the name of the name running mod. **Code Example:** local sCurrent = strupper(Game:GetCurrentModName());
`AddCommand (string, string, string)`	**Usage:** Adds a new command to the console. **Parameters:** string: The name of the command. string: The command itself. string: A help string than can be shown in the console with the '?'. **Return:** none **Code Example:** not used

CScriptObjectGame: C++ functions available in Lua script	
`EnableQuicksave (bool)`	**Usage:** Allows quick save or not. **Parameters:** bool: Enable quick save if != nil, disable it if nil. **Return:** none **Code Example:** not used