Engine scripts

Notice: Some of the functions below require the Amnesia 1.3 update. Steam copies should be automatically updated. Other copies can go here.

Main

The following functions are the main hps functions that the HPL2 engine looks to run on certain events - similar to the C++ int main() function.

```
void OnStart();
```

The function that runs when the map is loaded for the first time.

```
void OnEnter();
```

The function that runs whenever the player enters a map.

```
void OnLeave();
```

The function that runs when the player leaves a map.

```
void OnGameStart();
```

This function is found in the global.hps file and the inventory.hps file, and is run when the game is first started by the player (ie via “Start New Game”).

General

```
float RandFloat(float afMin, float afMax);
```

Generates a random float.

1. *afMin* - minimum value
2. *afMax* - maximum value

```
int RandInt(int alMin, int alMax);
```

Generates a random int. Note: the maximum value is *inclusive* - the RandInt() function may return this value.

1. *alMin* - minimum value
2. *alMax* - maximum value

```
bool StringContains(string& asString, string& asSubString);
```

Checks whether a string contains the specified string.

Example: searching for “hello” in “hello world” would return **true**.

1. *asString* - the string to check
2. *asSubString* - the string to search for

```
string& StringSub(string& asString, int alStart, int alCount);
```

Returns the substring in a string.
Example: in the string “frictional games rocks”, using 4 as *alStart* and 6 as *alCount* would return **“tional”**.

1. *asString* - the string
2. *alStart* - start position in the string
3. *alCount* - amount of characters

```
int StringToInt(string&in asString);
```

Requires 1.3

If possible, returns an integer converted from a string, else returns 0.

1. *asString* - String to convert.

```
float StringToFloat(string&in asString);
```

Requires 1.3

If possible, returns a float converted from a string, else returns 0.

1. *asString* - String to convert.

```
bool StringToBool(string&in asString);
```

Requires 1.3

If possible, returns a boolean converted from a string, else returns false.

1. *asString* - String to convert.

Mathematical Operations

```
float MathSin(float afX);
```

Requires 1.3

Returns the sine of the specified value.

1. *afX* - Value to operate.

```
float MathCos(float afX);
```

Requires 1.3

Returns the cosine of the specified value.

1. *afX* - Value to operate.

```
float MathTan(float afX);
```

Requires 1.3

Returns the tangent of the specified value.

1. *afX* - Value to operate.

```
float MathAsin(float afX);
```

Requires 1.3

Returns the arc sine of the specified value.

1. *afX* - Value to operate.

```
float MathAcos(float afX);
```

Requires 1.3

Returns the arc cosine of the specified value.

1. *afX* - Value to operate.

```
float MathAtan(float afX);
```

Requires 1.3

Returns the arc tangent of the specified value.

1. *afX* - Value to operate.

```
float MathAtan2(float afX, float afY);
```

Requires 1.3

Calculates and returns the arc tangent of the specified values.

1. *afX* - First value to operate.
2. *afY* - Second value to operate.

```
float MathSqrt(float afX);
```

Requires 1.3

Returns the square root of the specified value.

1. *afX* - Value to operate.

```
float MathPow(float afBase, float afExp);
```

Requires 1.3

Returns the value of afBase raised to the power of afExp.

1. *afBase* - The base value.
2. *afExp* - Value to calculate the base with.

```
float MathMin(float afA, float afB);
```

Requires 1.3

Returns the lowest value.

1. *afA* - First value.
2. *afB* - Second value.

```
float MathMax(float afA, float afB);
```

Requires 1.3

Returns the highest value.

1. *afA* - First value.
2. *afB* - Second value.

```
float MathClamp(float afX, float afMin, float afMax);
```

Requires 1.3

Returns afX clamped between afMin and afMax. If afX < afMin, returns afMin, and if afX > afMax, returns afMax.

1. *afX* - The value to clamp.
2. *afMin* - The minimum value to clamp afX with.
3. *afMax* - The maximum value to clamp afX with.

```
float MathAbs(float afX);
```

Requires 1.3

Returns the absolute value.

1. *afX* - Value to operate.

Debugging

```
void Print (string& asString);
```

Prints a string to the log file (`hpl.log`).

```
void AddDebugMessage(string& asString, bool abCheckForDuplicates);
```

Prints a string to the debug console.

1. *asString* - the string to print
2. *abCheckForDuplicates* - if true, the string won't be printed more than once on screen until it disappears

```
void ProgLog(string& asLevel, string& asMessage);
```

Prints an entry to the ProgLog (progression log).
ProgLog is a file created in Documents/Amnesia/main (or an FC folder if one is being used). It logs certain events, such us opening the menu or picking up the lantern, as well as the player's state (Health, Sanity, Oil, Tinderboxes, Coins), for the purpose of documenting a tester's playstyle.

This function allows to log custom messages.The messages in the ProgLog file are sorted by time elapsed since a map was loaded.

ProgLog has to be enabled for a player profile in `user_settings.cfg` before it starts working.

1. *asLevel* - can be “Low”, “Medium” or “High”. It's a tag which appears in each log entry, for event prioritising.
2. *asMessage* - The custom message to be printed to the log.

```
bool ScriptDebugOn();
```

Checks whether the debug mode is enabled.
See Setting up Development Environment to setup debug mode on your own computer.

Variables

Local

Local variables can be used throughout the same script file.

```
void SetLocalVarInt(string& asName, int alVal);
void AddLocalVarInt(string& asName, int alVal);
int GetLocalVarInt(string& asName);
```

```
void SetLocalVarFloat(string& asName, float afVal);
void AddLocalVarFloat(string& asName, float afVal);
float GetLocalVarFloat(string& asName);
```

```
void SetLocalVarString(string& asName, const string& asVal);
void AddLocalVarString(string& asName, string& asVal);
string& GetLocalVarString(string& asName);
```

Global

Global variables can be used throughout several maps and can be accessed by several script files.

```
void SetGlobalVarInt(string& asName, int alVal);
void AddGlobalVarInt(string& asName, int alVal);
int GetGlobalVarInt(string& asName);
```

```
void SetGlobalVarFloat(string& asName, float afVal);
void AddGlobalVarFloat(string& asName, float afVal);
float GetGlobalVarFloat(string& asName);
```

```
void SetGlobalVarString(string& asName, const string& asVal);
void AddGlobalVarString(string& asName, string& asVal);
string& GetGlobalVarString(string& asName);
```

Particle Systems

```
void PreloadParticleSystem(string& asPSFile);
```

Preloads a particle system.

1. *asPSFile* - The particle system file to load. Extension: .ps

```
void CreateParticleSystemAtEntity(string& asPSName, string& asPSFile,
string& asEntity, bool abSavePS);
```

Creates a particle system on an entity.

1. *asPSName* - internal name
2. *asPSFile* - the particle system to use + extension .ps
3. *asEntity* - the entity to create the particle system at
4. *abSavePS* - determines whether a particle system should “remember” its shown/hidden state, so that this state can be restored when the player revisits the level

```
void CreateParticleSystemAtEntityExt(string& asPSName, string& asPSFile,
string& asEntity, bool abSavePS,
float afR, float afG, float afB, float afA, bool abFadeAtDistance, float
afFadeMinEnd, float afFadeMinStart,
float afFadeMaxStart, float afFadeMaxEnd);
```

Creates a particle system on an entity, extended method with more options.

1. *asPSName* - internal name
2. *asPSFile* - the particle system to use + extension .ps
3. *asEntity* - the entity to create the particle system at
4. *abSavePS* - determines whether a particle system should “remember” its shown/hidden state, so that this state can be restored when the player revisits the level
5. *afR* - red value

6. *afG* - green value
7. *afB* - blue value
8. *afA* - alpha value
9. *abFadeAtDistance* - determines whether a particle system fades from a certain distance on
10. *afFadeMinEnd* - minimum distance at which the particle system stops fading
11. *afFadeMinStart* - minimum distance at which the particle system starts fading
12. *afFadeMaxStart* - maximum distance at which the particle system starts fading
13. *afFadeMaxEnd* - maximum distance at which the particle system stops fading

```
void DestroyParticleSystem(string& asName);
```

Destroys a particle system.

1. *asName* - The internal name of the particle system

Sounds & Music

```
void PreloadSound(string& asSoundFile);
```

Preloads a sound.

1. *asSoundFile* - The sound file to load. Extension: .snt

```
void PlaySoundAtEntity(string& asSoundName, string& asSoundFile, string&
asEntity, float afFadeTime, bool abSaveSound);
```

Creates a sound on an entity.

1. *asSoundName* - internal name
2. *asSoundFile* - the sound to use + extension .snt
3. *asEntity* - the entity to create the sound at, can be “Player”
4. *afFadeTime* - time in seconds the sound needs to fade. Avoids enemies hearing the sound if afFadeTime is at least 0.1f
5. *abSaveSound* - if `true`, a looping sound will “remember” its playback state (currently playing/stopped), and that state will be restored the next time the level is entered. If `true`, the sound is never attached to the entity! Note that saving should only be used on *looping sounds*!

```
void FadeInSound(string& asSoundName, float afFadeTime, bool abPlayStart);
```

Fades in a sound.

1. *asSoundName* - internal name
2. *afFadeTime* - time in seconds
3. *abPlayStart* - ?

```
void StopSound(string& asSoundName, float afFadeTime);
```

Fades out a sound.

1. *asSoundName* - internal name
2. *afFadeTime* - time in seconds, use 0 to immediatly stop the sound

```
void PlayMusic(string& asMusicFile, bool abLoop, float afVolume, float
afFadeTime, int alPrio, bool abResume);
```

Plays music.

1. *asMusicFile* - the music to play + extension .ogg
2. *abLoop* - determines whether a music track should loop
3. *afVolume* - volume of the music
4. *afFadeTime* - time in seconds until music reaches full volume
5. *alPrio* - priority of the music. Note that only the music with the highest priority can be heard! 0 - lowest, 1 - higher, etc.
6. *abResume* - if `true`, playback will be continued from where the track stopped after the call to StopMusic(); if `false`, the track will be restarted.

```
void StopMusic(float afFadeTime, int alPrio);
```

Stops music.

1. *afFadeTime* - time in seconds until music stops
2. *alPrio* - the priority of the music that should stop

```
void FadeGlobalSoundVolume(float afDestVolume, float afTime);
```

Influences the global sound volume, that means everything you can hear **from the world**. This does not affect music of GUI sounds.

1. *afDestVolume* - desired volume
2. *afTime* - time in seconds until volume reaches desired volume

```
void FadeGlobalSoundSpeed(float afDestSpeed, float afTime);
```

Influences the global sound speed.

1. *afDestSpeed* - desired speed
2. *afTime* - time in seconds until volume reaches desired speed

Lights

```
void SetLightVisible(string& asLightName, bool abVisible);
```

Enables/disables lights.

1. *asLightName* - internal name
2. *abVisible* - determines the state of the light

```
void FadeLightTo(string& asLightName, float afR, float afG, float afB, float
afA, float afRadius, float afTime);
```

Changes the properties of a light.

1. *asLightName* - internal name

2. *afR* - red value
3. *afG* - green value
4. *afB* - blue value
5. *afA* - alpha value
6. *afRadius* - radius of the light. -1 means keeping the radius
7. *afTime* - time in seconds until change is done

```
void SetLightFlickerActive(string& asLightName, bool abActive);
```

Activates flickering on a light.

1. *asLightName* - The internal light name
2. *abActive* - true = active, false = inactive

Game scripts

General

```
void StartCredits(string& asMusic, bool abLoopMusic, string& asTextCat,
string& asTextEntry, int alEndNum);
```

Starts the end credits screen.

1. *asMusic* - the music to play (including .ogg)
2. *abLoopMusic* - determines whether the music should loop
3. *asTextCat* - the category to be used in the .lang file
4. *asTextEntry* - the entry in the .lang file
5. *alEndNum* - Amnesia has 3 different endings and displays a code at the bottom. Determines which code is displayed. 0-2 will display codes, any other integer will not.

```
void StartDemoEnd();
```

Starts the end images that are used in the demo, images are named “demo_end01.jpg”, increase the number for each image you want to use. (NEEDS VERIFICATION)

```
void AutoSave();
```

Save the game to the auto save.

```
void CheckPoint (string& asName, string& asStartPos, string& asCallback,
string& asDeathHintCat, string& asDeathHintEntry);
```

Sets a checkpoint at which the player will respawn in case he dies.
Callback syntax: `void MyFunc(string &in asName, int alCount)`
Count is 0 on the first checkpoint load!

1. *asName* - the internal name
2. *asStartPos* - the name of the StartPos in the editor
3. *asCallback* - the function to call when the player dies/respawns

4. *asDeathHintCat* - the category of the death hint message to be used in the .lang file
5. *asDeathHintEntry* - the entry in the .lang file

```
void ChangeMap(string& asMapName, string& asStartPos, string& asStartSound,
string& asEndSound);
```

Immediatly loads another map.

1. *asMapName* - the file to load
2. *asStartPos* - the name of the StartPos on the next map
3. *asStartSound* - the sound that is played when the change starts
4. *asEndSound* - the sound that is played when the new map is loaded

```
void ClearSavedMaps();
```

Clears the “history” of the save, useful to do when you know the player will not be able to go back anymore. Makes the next save much smaller in size.

```
void CreateDataCache();
void DestroyDataCache();
```

This caches all current textures and models and they are not released until destroy is called. If there is already cached data it is destroyed.

```
void SetMapDisplayNameEntry(string& asNameEntry);
```

Sets the map name shown in save file names. If none is set NULL is assumed.

1. *asNameEntry* - the entry to display, category must be “Levels”!

```
void SetSkyBoxActive(bool abActive);
```

Enables/Disables the skybox.

1. *abActive* - true = active, false = inactive

```
void SetSkyBoxTexture(string& asTexture);
```

Sets the texture of the skybox.

1. *asTexture* - The texture file to set. Extension: .dds

```
void SetSkyBoxColor(float afR, float afG, float afB, float afA);
```

Sets the solid color of the skybox rather than a texture.

1. *afR* - red value
2. *afG* - green value
3. *afB* - blue value
4. *afA* - alpha value

```
void SetFogActive(bool abActive);
```

Enables/Disables the global fog.

1. *abActive* - true = active, false = inactive

```
void SetFogColor(float afR, float afG, float afB, float afA);
```

Sets the color to use for the global fog.

1. *afR* - red value
2. *afG* - green value
3. *afB* - blue value
4. *afA* - alpha value

```
void SetFogProperties(float afStart, float afEnd, float afFalloffExp, bool
abCulling);
```

Sets the properties for the global fog.

1. *afStart* - how many meters from the camera should the fog begin
2. *afEnd* - how many meters from the camera should the fog reach full thickness
3. *afFalloffExp* - the amount by which the thinkness increases
4. *abCulling* - whether occlusion culling is active for the fog; this prevents objects behind the fog from being loaded

```
void SetupLoadScreen(string&asTextCat, string&asTextEntry, int alRandomNum,
string&asImageFile);
```

Determines which loading screen will be shown when changing maps.

1. *asTextCat* - the category of the loading text in the .lang file to be shown on the loading screen
2. *asTextEntry* - the entry in the .lang file
3. *alRandomNum* - if greater 1, then it will randomize between 1 and alRandom for each LoadScreen giving entry the suffix XX (eg 01). If < =1 then no suffix is added
4. *asImageFile* - the image to be shown (optional)

Game Timer

```
void AddTimer(string& asName, float afTime, string& asFunction);
```

Creates a timer which calls a function when it expires.
Callback syntax: `void MyFunc(string &in asTimer)`

1. *asName* - the name of the timer
2. *afTime* - time in seconds
3. *asFunction* - the function to call

```
void RemoveTimer(string& asName);
```

Removes a timer, no matter how much time is left.

1. *asName* - the internal name of the timer.

```
float GetTimerTimeLeft(string& asName);
```

Returns the time left on a timer.

1. *asName* - the internal name of the timer.

Screen Effects

```
void FadeOut(float afTime);
```

Fades the screen to black.

afTime - time in seconds until the screen is completly black

```
void FadeIn(float afTime);
```

Fades the screen back to normal.

afTime - time in seconds until the screen back to normal

```
void FadeImageTrailTo(float afAmount, float afSpeed);
```

Applies the image trail effect to the screen.

afAmount - intensity (default: 0)
afSpeed - time in seconds until full effect

```
void FadeSepiaColorTo(float afAmount, float afSpeed);
```

Makes the screen go dark red.

afAmount - intensity (default: 0)
afSpeed - time in seconds until full effect

```
void FadeRadialBlurTo(float afSize, float afSpeed);
```

Applies radial blur effects to the screen.

afSize - intensity (default: 0)
afSpeed - time in seconds until full effect

```
void SetRadialBlurStartDist(float afStartDist);
```

Determines at which distance the radial blur effects appear.

afStartDist - the distance at which the effect starts

```
void StartEffectFlash(float afFadeIn, float afWhite, float afFadeOut);
```

Fades the screen to white.

afFadeIn - time in seconds until screen is white
afWhite - determines to which percentage the screen fades to white (1.0 = completly white)
afFadeOut - time in seconds until screen is back to normal again

```
void StartEffectEmotionFlash(string& asTextCat, string& asTextEntry, string&
asSound);
```

Fades the screen to white and shows a text message.

asTextCat - the category in the .lang file
asTextEntry - the text entry in the .lang file
asSound - the sound to play while fading

```
void AddEffectVoice(string& asVoiceFile, string& asEffectFile, string&
asTextCat, string& asTextEntry,
bool abUsePosition, string& asPosEntity, float afMinDistance, float
afMaxDistance);
```

This adds a voice and an effect to be played. It is okay to call this many times in order to play many voices in a row. The EffectVoiceOverCallback is not called until ALL voices have finished.

asVoiceFile - the voice to play
asEffectFile - the effect to play
asTextCat - the category in the .lang file
asTextEntry - the text entry in the .lang file
abUsePosition - plays using 3D from the entity, or without 3D
asPosEntity - the entity at which the effect appears
afMinDistance - minimum distance to see the effect
afMaxDistance - maximum distance to see the effect

```
void StopAllEffectVoices(float afFadeOutTime);
```

Stops all voices and calls the EffectVoiceOverCallback.

```
bool GetEffectVoiceActive();
```

Checks whether EffectVoices are still active.

```
void SetEffectVoiceOverCallback(string& asFunc);
```

Sets the function to be called when the EffectVoices are finished.
Callback syntax: **void MyFunc()**

```
bool GetFlashbackIsActive();
```

Checks whether a flashback is still in effect.

```
void StartPlayerSpawnPS(string& asSPSFile);
void StopPlayerSpawnPS();
```

Continuously spawn regular particle systems (`.ps`) around the player. Particles created by this script carry over from map to map.

asSPSFile - the `.sps` file to use. Exemplary `.sps` files are located in the `/misc` folder in the main game directory.

Custom `.sps` files can be created by hand in a text editor (see existing ones and mimic how those are written).
Since `StopPlayerSpawnPS()` doesn't seem to work, to stop an SPS you must create an `.sps` file with an empty particle field field and override the old SPS by calling `StartPlayerSpawnPS` again.

```
void PlayGuiSound(string& asSoundFile, float afVolume);
```

Plays a sound, not using 3D.

asSoundFile - the sound to play (extension is .snt)
afVolume - the volume of the sound

```
void StartScreenShake(float afAmount, float afTime, float afFadeInTime,
float afFadeOutTime);
```

Shakes the screen.

afAmount - intensity of the shake
afTime - duration of the shake
afFadeInTime - time in seconds until full intensity is reached
afFadeOutTime - time until screen is back to normal

```
void SetInDarknessEffectsActive(bool abX);
```

Requires 1.3

Enables/disables the sanity drain and night vision effects while in the darkness.

bool abX - Enable/disable effects.

Insanity

```
void SetInsanitySetEnabled(string& asSet, bool abX);
```

Determines which InsanitySets are enabled.

asSet - the set
abX - enabled or not

```
void StartInsanityEvent(string &in asEventName);
```

Requires 1.3

Starts a specified insanity event.

asEventName - Insanity event to play.

```
void StartRandomInsanityEvent();
```

Starts a random insanity event from the available sets.

```
void StopCurrentInsanityEvent();
```

Requires 1.3

Stops the currently playing insanity event.

```
void InsanityEventIsActive();
```

Checks whether an insanity event is currently in effect... Or so it was supposed to be, but as it doesn't return a value, we can never know

Player

Note that the player's maximum health and sanity is 100.

```
void SetPlayerActive(bool abActive);
```

Enabled/Disable player controlled movement.

```
void ChangePlayerStateToNormal();
```

Sets certain effects back to normal. It can for example make the player drop an item.

```
void SetPlayerCrouching(bool abCrouch);
```

Forces the player to crouch.

```
void AddPlayerBodyForce(float afX, float afY, float afZ, bool
abUseLocalCoords);
```

Pushes the player into a certain direction. Note that you need values above ~2000 to see any effects.

afX - amount along the X-axis
afY - amount along the Y-axis
afZ - amount along the Z-axis
abUseLocalCoords - If true, axes are based on where the player is facing, not the world.

```
void ShowPlayerCrossHairIcons(bool abX);
```

Enables/Disables the icons when a player has something in focus.

```
void SetPlayerSanity(float afSanity);
void AddPlayerSanity(float afSanity);
```

```
float GetPlayerSanity();
```

Modifies/returns the sanity of the player.

```
void SetPlayerHealth(float afHealth);
void AddPlayerHealth(float afHealth);
float GetPlayerHealth();
```

Modifies/returns the health of the player.

```
void SetPlayerLampOil(float afOil);
void AddPlayerLampOil(float afOil);
float GetPlayerLampOil();
```

Modifies/returns the lamp oil of the player.

```
float GetPlayerSpeed();
float GetPlayerYSpeed();
```

Returns the current speed of the player.

```
void SetSanityDrainDisabled(bool abX);
```

Enables/Disables sanity drain from darkness, monsters, etc.

```
void GiveSanityBoost();
void GiveSanityBoostSmall();
```

Boosts the player's sanity by a fixed amount.

```
void GiveSanityDamage(float afAmount, bool abUseEffect);
```

Reduces the sanity of the player.

afAmount - amount of sanity damage done
abUseEffect - determines whether an effect is played when the sanity damage is dealt

```
void GivePlayerDamage(float afAmount, string& asType, bool abSpinHead, bool
abLethal);
```

Reduces the health of the player.

afAmount - amount of damage done to health
asType - plays a certain effect on the screen when the damage is dealt (BloodSplat, Claws or Slash)
abSpinHead - changes the camera view when damage is dealt
abLethal - set to true if player can die from given damage

```
void FadePlayerFOVMulTo(float afX, float afSpeed);
```

Changes the field of view of the player. A shorter FOV will create a zoom effect.

afX - multiplier of default FOV (1 is default)
afSpeed - the speed of change between FOV's

```
void FadePlayerAspectMulTo(float afX, float afSpeed);
```

Changes the aspect ratio of the player. Basically stretches or narrows the screen horizontally.

afX - multiplier of default aspect (default is 1)
afSpeed - the speed of change between FOV's

```
void FadePlayerRollTo(float afX, float afSpeedMul, float afMaxSpeed);
```

Rotates the position of the camera on the player's body.

afX - angle of rotation of head, positive being counter-clockwise
afSpeedMul - speed (possibly acceleration) multiplier of the rotation (default 1, which is really slow)
afMaxSpeed - maximum speed of rotation

```
void MovePlayerHeadPos(float afX, float afY, float afZ, float afSpeed, float
afSlowDownDist);
```

Changes the position of the camera on the player's body.

afX - amount along the X-axis
afY - amount along the Y-axis
afZ - amount along the Z-axis
afSpeed - speed at which the change happens
afSlowDownDist - distance at which to start slowing down (prevents the head from abruptly stopping)

```
void StartPlayerLookAt(string& asEntityName, float afSpeedMul, float
afMaxSpeed, string& asAtTargetCallback);
void StopPlayerLookAt();
```

Forces the player to look at a certain entity until StopPlayerLookAt is used.

asEntityName - the entity to look at
afSpeedMul - how fast should the player look at the entity
afMaxSpeed - maximum speed allowed
asAtTargetCallback - function to call when player looks at target

```
void SetPlayerMoveSpeedMul(float afMul);
void SetPlayerRunSpeedMul(float afMul);
void SetPlayerLookSpeedMul(float afMul);
```

Changes the player's move/run/look speed. Default is 1.

```
void SetPlayerJumpForceMul(float afMul);
```

Requires 1.3

Changes the player's jump multiplier. Higher values = higher jumps. Default is 1.

```
void SetPlayerJumpDisabled(bool abX);
void SetPlayerCrouchDisabled(bool abX);
```

Enables/Disables the player's ability to jump/crouch.

```
void TeleportPlayer(string& asStartPosName);
```

Instantly teleports the player to the target StartPos.

```
void SetLanternActive(bool abX, bool abUseEffects);
```

Makes the player use his lantern.

```
bool GetLanternActive();
```

Checks whether the player currently uses his lantern.

```
void SetLanternDisabled(bool abX);
```

Enables/Disables the player's ability to use his lantern.

```
void SetLanternLitCallback(string& asCallback);
```

Sets the function to call when the player uses his lantern.
Callback syntax: **MyFunc(bool abLit)**

```
void SetMessage(string& asTextCategory, string& asTextEntry, float afTime);
```

Displays a message on the screen.

asTextCategory - the category in the .lang file
asTextEntry - the entry in the .lang file
afTime - determines how long the message is displayed. If time is < =0 then the life time is calculated based on string length.

```
void SetDeathHint(string& asTextCategory, string& asTextEntry);
```

Sets the message that appears when the player dies.

asTextCategory - the category in the .lang file
asTextEntry - the entry in the .lang file

```
void DisableDeathStartSound();
```

Disables the death sound when the player dies. This must be called directly before player is killed! The variable as soon as player dies too.

```
void MovePlayerForward(float afAmount)
```

“REQUIRES THE 1.2 PATCH: JUSTINE” Moves the player forward. It needs to be called in a timer that

updates 60 times / second.

```
void SetPlayerFallDamageDisabled(bool abX);
```

Requires 1.3

Enables/disables the player's ability to take fall damage.

```
void SetPlayerPos(float afX, float afY, float afZ);
```

Requires 1.3

Sets the player's position within the level.

afX - X co-ordinate position.
afY - Y co-ordinate position.
afZ - Z co-ordinate position.

```
float GetPlayerPosX();
float GetPlayerPosY();
float GetPlayerPosZ();
```

Requires 1.3

Returns the player's position within the level on the specified axis.

Journal

```
void AddNote(string& asNameAndTextEntry, string& asImage);
```

Adds a note to the player's journal.

asNameAndTextEntry - entries in the .lang file. Must end with _Name and _Text and be in category “Journal”!
asImage - the background image to be used

```
void AddDiary(string& asNameAndTextEntry, string& asImage);
```

Adds a diary to the player's journal.

asNameAndTextEntry - entries in the .lang file. Must end with _NameX and _TextY whereas X and Y are numbers of the parts (_Name1: first diary, _Text1: first page) and be in category “Journal”!
asImage - the background image to be used

```
void ReturnOpenJournal(bool abOpenJournal);
```

Only called in the pickup diary callback! If true the journal displays the entry else not.

Quests

```
void AddQuest(string& asName, string& asNameAndTextEntry);
```

Adds a quest to the player's journal.

asName - the internal name to be used
asNameAndTextEntry - entry in the .lang file. Must start with “Quest_<texthere>_Text”, and be in category “Journal”!

```
void CompleteQuest(string& asName, string& asNameAndTextEntry);
```

Completes a quest.

asName - the internal name of the quest
asNameAndTextEntry - entry in the .lang file. Must start with ” Quest_<texthere>_Text ”, and be in category “Journal”!

```
bool QuestIsCompleted(string& asName);
bool QuestIsAdded(string& asName);
```

Checks whether a quest is completed/added.

```
void SetNumberOfQuestsInMap(int alNumberOfQuests);
```

Sets the number of quests in the map.

alNumberOfQuests - Amount of Quests

```
void GiveHint (string& asName, string& asMessageCat, string& asMessageEntry,
float afTimeShown);
```

Displays a hint on the player's screen.

asName - the internal name
asMessageCat - the category in the .lang file
asMessageEntry - the entry in the .lang file
afTimeShown - time in seconds until the message disappears. If time is ⇐ 0 then the life time is calculated based on string length.

```
void RemoveHint (string& asName);
void BlockHint (string& asName);
void UnBlockHint (string& asName);
```

Removes\Blocks\Unblocks a hint.

Inventory

```
void ExitInventory();
```

Exits the inventory by force.

```
void SetInventoryDisabled(bool abX);
```

Disables the player's ability to open his inventory.

```
void SetInventoryMessage(string& asTextCategory, string& asTextEntry, float
afTime);
```

Adds a message at the bottom of the inventory screen.

asTextCategory - the category in the .lang file
asTextEntry - the entry in the .lang file
afTime - time in seconds until the message disappears. If life time is ⇐ 0 then the life time is calculated based on string length.

```
void GiveItem(string& asName, string& asType, string& asSubTypeName, string&
asImageName, float afAmount);
```

Adds an item to the inventory of the player. Note that the item does not have to exist as entity in the world to be able to do this.

asName - internal name
asType - item to give
asSubTypeName - item name for .lang file
asImageName - For exemple: void GiveItem(string& asName, string& asType, “chemical_container_full”, “chemical_container_full.tga” , float afAmount); The image is from *<nowiki> <nowiki> <nowiki>\ </nowiki> </nowiki> </nowiki>* graphics\Item\chemical_container_full.tga : is the image which will appear in Inventory - img155.imageshack.us/img155/6871/20806785.jpg

afAmount - amount to give

```
void RemoveItem(string& asName);
```

Removes an item from the player's inventory.

```
bool HasItem(string& asName);
```

Checks whether the player has an item in his inventory.

```
void GiveItemFromFile(string& asName, string& asFileName);
```

Adds a single item to the player's inventory. This is meant to be used for debug mostly as it creates the actual item and then destroys it.

asName - internal name
asFileName - item to give + extension (.ent)

```
void AddCombineCallback(string& asName, string& asItemA, string& asItemB,
```

```
string& asFunction, bool abAutoRemove);
```

Allows the player to combine items in his inventory.
Callback syntax: **void MyFunc(string &in asItemA, string &in asItemB)**

asName - internal name for the callback
asItemA - internal name of first item
asItemB - internal name of second item
asFunction - the function to call
abAutoRemove - determines whether the callback should be removed when the items are combined

```
void RemoveCombineCallback(string& asName);
```

Removes a combine callback.
asName - the internal name of the callback to be removed (as specified in AddCombineCallback)

```
void AddUseItemCallback(string& asName, string& asItem, string& asEntity,
string& asFunction, bool abAutoDestroy);
```

Allows the player to use items on the world.
Callback syntax: **void MyFunc(string &in asItem, string &in asEntity)**

asName - internal name
asItem - internal name of the item
asEntity - entity to be able to use the item on
asFunction - function to call
abAutoDestroy - determines whether the item is destroyed when used

```
void RemoveUseItemCallback(string& asName);
```

Removes an item callback.

Entities

General

```
void SetEntityActive(string& asName, bool abActive);
```

Activates/deactivates an entity.

```
void SetEntityVisible(string &in asName, bool abVisible);
```

Requires 1.3

Activates/deactivates an entity's visual mesh. The collision body remains.

asName - Name of the entity.
abActive - Activate/deactivate mesh.

```
bool GetEntityExists(string& asName);
```

Checks whether an entity exists.

```
void SetEntityCustomFocusCrossHair(string& asName, string& asCrossHair);
```

Changes the crosshair that is used when focusing an entity.

asName - internal name
asCrossHair - desired crosshair, can be: Default (uses default), Grab, Push, Ignite, Pick, LevelDoor, Ladder

```
void CreateEntityAtArea(string& asEntityName, string& asEntityFile, string&
asAreaName, bool abFullGameSave);
```

Creates an entity at an area. When creating an enemy though, it cannot chase properly along PathNodes(using for example ShowEnemyPlayerPosition).

asEntityName - internal name
asEntityFile - entity to be used extension .ent
asAreaName - the area to create the entity at
abFullGameSave - determines whether an entity “remembers” its state

```
void ReplaceEntity(string &in asName, string &in asBodyName, string &in
asNewEntityName, string &in asNewEntityFile, bool abFullGameSave);
```

Requires 1.3

Removes an entity and places a new one in its place.

asName - Name of the entity to replace.
asBodyName - Name of the body of the entity to place the new entity at. If empty the first body is used (might be buggy, recommended to name a body anyway).
asNewEntityName - Name of the new entity.
asNewEntityFile - Name of the new entity file. Extension .ent.
abFullGameSave - Whether ALL properties of this entity should be saved throughout levels.

```
void PlaceEntityAtEntity(string &in asName, string &in asTargetEntity,
string &in asTargetBodyName, bool abUseRotation);
```

Requires 1.3

Places an entity at the position of another entity. Does not work for enemies, use TeleportEnemyToEntity instead.

asName - Name of the entity to place.
asTargetEntity - Name of the other entity to place the first entity at.
asTargetBodyName - Name of the body of the entity to place the first entity at. If empty the first body is used (might be buggy, recommended to name a body anyway).
abUseRotation - Whether the entity should be rotated like the target entity.

```
void SetEntityPos(string &in asName, float afX, float afY, float afZ);
```

Requires 1.3

Moves an entity to a position in the level.

asName - Name of the entity to move.
afX - X co-ordinate position.
afY - Y co-ordinate position.
afZ - Z co-ordinate position.

```
float GetEntityPosX(string &in asName);
float GetEntityPosY(string &in asName);
float GetEntityPosZ(string &in asName);
```

Requires 1.3

Returns an entity's position in the level on the specified axis.

asName - Name of the entity.

```
void SetEntityPlayerLookAtCallback(string& asName, string& asCallback, bool
abRemoveWhenLookedAt);
```

Calls a function when the player looks at a certain entity.
Callback syntax: **void MyFunc(string &in asEntity, int alState)**
alState: 1 = looking, -1 = not looking

asName - internal name
asCallback - function to call
abRemoveWhenLookedAt - determines whether the callback should be removed when the player looked at the entity

```
void SetEntityPlayerInteractCallback(string& asName, string& asCallback,
bool abRemoveOnInteraction);
```

Calls a function when the player interacts with a certain entity.
Callback syntax: **void MyFunc(string &in asEntity)**

asName - internal name
asCallback - function to call
abRemoveOnInteraction - determines whether the callback should be removed when the player interacts with the entity

```
void SetEntityCallbackFunc(string& asName, string& asCallback);
```

Calls a function when the player interacts with a certain entity.
Callback syntax: **void MyFunc(string &in asEntity, string &in type)**
Type depends on entity type and includes: “OnPickup”, “Break”, “OnIgnite”, etc

```
void SetEntityConnectionStateChangeCallback(string& asName, string&
```

```
asCallback);
```

A callback called when ever the connection state changes (button being switched on, lever switched, etc).
Callback syntax: **void Func(string &in asEntity, int alState)**
alState: -1 = off, 0 = between, 1 = on

```
void SetEntityInteractionDisabled(string& asName, bool abDisabled);
```

Disallows interaction with an entity.

```
void BreakJoint (string& asName);
```

Breaks a joint. Do not use this on joints in SwingDoors, Levers, Wheels, etc. where the joint is part of an interaction. That will make the game crash.

```
void AddEntityCollideCallback(string& asParentName, string& asChildName,
string& asFunction, bool abDeleteOnCollide, int alStates);
```

Calls a function when two entites collide.
Callback syntax: **void MyFunc(string &in asParent, string &in asChild, int alState)**
alState: 1 = enter, -1 = leave

asParentName - internal name of main object
asChildName - internal name of object that collides with main object (asterix (*) NOT supported!)
asFunction - function to call
abDeleteOnCollide - determines whether the callback after it was called
alStates - 1 = only enter, -1 = only leave, 0 = both

```
void RemoveEntityCollideCallback(string& asParentName, string& asChildName);
```

Removes an EntityCollideCallback. Asterix (*) not supported in *asChildName*.

```
bool GetEntitiesCollide(string& asEntityA, string& asEntityB);
```

Checks whether two entites collide. This function does NOT support asterix (*) or “Player”!

```
void SetBodyMass(string &in asName, float afMass);
```

Requires 1.3

Sets the mass of an entity's body.

asName - Name of the body of an entity. The body name of an entity is EntityName_BodyName.
afMass - The mass to set.

```
float GetBodyMass(string &in asName);
```

Requires 1.3

Gets the mass of an entity's body.

asName - Name of the body of an entity. The body name of an entity is EntityName_BodyName.
afMass - The mass to get.

Props

```
void SetPropEffectActive(string& asName, bool abActive, bool
abFadeAndPlaySounds);
```

Can be used on coal to give it the black color it should have.

```
void SetPropActiveAndFade(string& asName, bool abActive, float afFadeTime);
```

Activates/deactivates a prop.

asName - internal name
abActive - nothing to add
afFadeTime - time in seconds until prop fully fades

```
void SetPropStaticPhysics(string& asName, bool abX);
```

Activates/deactivates the physics of a prop. Setting as true will make entities static in midair.

```
bool GetPropIsInteractedWith(string& asName);
```

Checks whether a prop is interacted with.

```
void RotatePropToSpeed(string& asName, float afAcc, float afGoalSpeed, float
afAxisX, float afAxisY, float afAxisZ, bool abResetSpeed, string&
asOffsetArea);
```

Rotates the prop up to a set speed.

asName - internal name
afAcc - acceleration
afGoalSpeed - desired speed
afAxisX - rotation around X axis
afAxisY - rotation around Y axis
afAxisZ - rotation around Z axis
abResetSpeed - determines whether the speed is resetted after goal speed is reached
asOffsetArea - the area to rotate around, if empty, then the center of the body is used Note: The entity you want to rotate MUST be a “StaticObject” entity!

```
void StopPropMovement(string& asName);
```

Stops all movement of a prop.

```
void AddAttachedPropToProp(string& asPropName, string& asAttachName, string&
asAttachFile, float afPosX, float afPosY, float afPosZ, float afRotX, float
afRotY, float afRotZ);
```

Attaches a prop to another prop.

a*sPropName*- the prop to attach another prop at
asAttachName - internal name of the prop that gets attached
asAttachFile - the prop that gets attached extension .ent
afPosX - X position of the attach from the prop
afPosY - Y position of the attach from the prop
afPosZ - Z position of the attach from the prop
afRotX - rotation around X axis of the attach
afRotY - rotation around Y axis of the attach
afRotZ - rotation around ZX axis of the attach Note: for the purposes of “AddEntityCollideCallback”, attached props will not call the callback function if they collide with a “static_object” or a “StaticProp” entity type!

Bug: *afRotZ* is used for both the ZX rotation and the Z position of the attached prop. Unwanted rotation can be avoided by using:
AddAttachedPropToProp(asPropName,asAttachName,asAttachFile,afPosX,afPosY,0,afPosZ,90.0f,afPosZ)

Bug: Attaching a breakable prop to a physically active prop, and then breaking the attached prop, will cause the game to crash should the parent object be moved or reset.

```
void AttachPropToProp(string& asPropName, string& asAttachName, string& asAttachFile, float afPosX, float afPosY, float afPosZ, float afRotX, float afRotY, float afRotZ);
```

Requires 1.3

Attaches a prop to another prop. Fixed version of AddAttachedPropToProp.

asPropName - the prop to attach another prop at
asAttachName - internal name of the prop that gets attached
asAttachFile - the prop that gets attached extension .ent
afPosX - X position of the attach from the prop
afPosY - Y position of the attach from the prop
afPosZ - Z position of the attach from the prop
afRotX - rotation around X axis of the attach
afRotY - rotation around Y axis of the attach
afRotZ - rotation around ZX axis of the attach Note: for the purposes of “AddEntityCollideCallback”, attached props will not call the callback function if they collide with a “static_object” or a “StaticProp” entity type!

```
void RemoveAttachedPropFromProp(string& asPropName, string& asAttachName);
```

Detaches a prop from a prop.

```
void SetPropHealth(string& asName, float afHealth);
void AddPropHealth(string& asName, float afHealth);
float GetPropHealth(string& asName);
```

Modifies/returns the health of a prop.

```
void ResetProp(string& asName);
```

Resets a prop's state to the original one when the map was loaded.

```
void PlayPropAnimation(string& asProp, string& asAnimation, float
afFadeTime, bool abLoop, string& asCallback);
```

Makes the prop play an animation and calls a function.
Callback syntax: **void MyFunc(string &in asProp)**

asProp - internal name of the prop
asAnimation - animation to play
afFadeTime - ?
abLoop - determines whether the animation loops
asCallback - function to call

```
void AddPropForce(string& asName, float afX, float afY, float afZ, string&
asCoordSystem);
void AddPropImpulse(string& asName, float afX, float afY, float afZ, string&
asCoordSystem);
void AddBodyForce(string& asName, float afX, float afY, float afZ, string&
asCoordSystem);
void AddBodyImpulse(string& asName, float afX, float afY, float afZ, string&
asCoordSystem);
```

These functions push objects. Note that rather high values are needed when applying *forces* (on the order of ~100 (weak) to ~10000 (strong)), but not impulses (values less than 10 can be appropriate). Forces are external influences, and will have different effect depending on the mass of the object they are being applied to; impulses disregard mass, and can cause objects to break, as if hit. A “Body” is a physics-related helper object, to which a force or an impulse can be applied. Entities can consist of several bodies, interconnected in various ways (you can create/examine bodies in the model editor).

asName - the object to push; for bodies, use this format: “*entityName_bodyName*”
afX - magnitude along the X-axis
afY - magnitude along the Y-axis
afZ - magnitude along the Z-axis
asCoordSystem - determines which coordinate system is used, usually “world” All of these functions are *additive* - when called consecutively, for each call, the vectors defined by (afX, afY, afZ) will be added together, and a resultant force/impulse will be calculated *before* any physics simulation is applied to the target object.

Connections

```
void InteractConnectPropWithRope(string& asName, string& asPropName, string&
asRopeName, bool abInteractOnly, float afSpeedMul, float afToMinSpeed, float
afToMaxSpeed, bool abInvert, int alStatesUsed);
```

Connects a prop with the movement of a rope (ie. turn wheel to move rope).

asName - connection name
asPropName - name of prop
asRopeName - name of rope
abInteractOnly - ?
afSpeedMul - speed multiplier of how quickly the rope moves
afToMinSpeed - the slowest the rope will move when moving the prop
afToMaxSpeed - the fastest the rope will move when moving the prop
abInvert - whether to invert the direction the rope moves
alStatesUsed - which states of the prop can interact with the rope?

```
void InteractConnectPropWithMoveObject(string& asName, string& asPropName,
string& asMoveObjectName, bool abInteractOnly, bool abInvert, int
alStatesUsed);
```

This one should only be used if there must be an exact correspondance to prope “amount” and the moveobject open amount. It is best used for Wheel-door connections!

```
void ConnectEntities(string& asName, string& asMainEntity, string&
asConnectEntity, bool abInvertStateSent, int alStatesUsed, string&
asCallbackFunc);
```

Callback syntax: **void MyFunc(string &in asConnectionName, string &in asMainEntity, string &in asConnectEntity, int alState)**
State is what is sent to connection entity and will be inverted if abInvertStateSent = true!

Lamps

```
void SetLampLit(string& asName, bool abLit, bool abEffects);
```

(Un)lits a lamp.

asName - Name of the lamp
abLit - Set true if you want the lamp to be lit, set to false if you want the lamp to be unlit
abEffects - If you want to have the lamp fade in/out when it gets (un)lit

Doors

```
void SetSwingDoorLocked(string& asName, bool abLocked, bool abEffects);
void SetSwingDoorClosed(string& asName, bool abClosed, bool abEffects);
```

Locks/closes a swing door.

```
bool GetSwingDoorLocked(string& asName);
bool GetSwingDoorClosed(string& asName);
```

Checks whether a swing door is locked/closed.

```
void SetSwingDoorDisableAutoClose(string& asName, bool abDisableAutoClose);
```

Deactivates the “auto-close” when a door is nearly closed.

```
int GetSwingDoorState(string& asName);
```

Returns an integer depending on how far the door is opened.
-1 = angle is close to 0°, 1 = angle is 70% or higher of max, 0 = inbetween -1 and 1.

```
void SetLevelDoorLocked(string& asName, bool abLocked);
```

Locks a level door. Note that level doors are NOT swing doors.

```
void SetLevelDoorLockedSound(string& asName, string& asSound);
```

Determines which sound is played when interacting with a locked level door.

```
void SetLevelDoorLockedText(string& asName, string& asTextCat, string&
asTextEntry);
```

Displays a message when interacting with a locked level door.

asName - internal name
asTextCat - the category in the .lang file
asTextEntry - the entry in the .lang file

```
void SetMoveObjectState(string& asName, float afState);
```

Moves an object to a certain state.

asName - internal name
afState - state of the object, 0 = closed, 1 = open, values inbetween (and above, for example, the bridge_metal_vert) are valid too!

```
void SetMoveObjectStateExt(string& asName, float afState, float afAcc, float
afMaxSpeed, float afSlowdownDist, bool abResetSpeed);
```

Moves an object to a certain state, extended method.

asName - internal name
afState - state of the object, 0 = closed, 1 = open, values inbetween are valid too!
afAcc - acceleration
afMaxSpeed - maximum speed
afSlowdownDist - Distance to the target state before decceleration occurs.
abResetSpeed - Set to True if the prop's speed should be reset before performing the movement, else the prop will accelerate from it's current speed to afMaxSpeed.

Levers, wheels and buttons

```
void SetPropObjectStuckState(string& asName, int alState);
void SetWheelStuckState(string& asName, int alState, bool abEffects);
void SetLeverStuckState(string& asName, int alState, bool abEffects);
```

Makes a prop\wheel\lever stuck in a certain state.

asName - internal name
alState - 0 = not stuck, 1 = at max, -1 = at min
abEffects - use effects

```
void SetWheelAngle(string& asName, float afAngle, bool abAutoMove);
```

Moves a wheel to a certain angle.

asName - internal name
afAngle - angle
abAutoMove - determines whether the wheel should move on its own

```
void SetWheelInteractionDisablesStuck(string& asName, bool abX);
void SetLeverInteractionDisablesStuck(string& asName, bool abX);
```

Allows the player to make a wheel/lever unstuck when interacted with.

```
int GetLeverState(string& asName);
```

Returns the state of the lever.
0 = not stuck, 1 = at max, -1 = at min

```
void SetMultiSliderStuckState(string& asName, int alStuckState, bool
abEffects);
```

Makes a MultiSlider stuck in a certain state.

```
void SetMultiSliderCallback(string& asName, string& asCallback);
```

Calls a function when state changes.
Callback syntax: **void MyFunc(string &in asEntity, int alState)**

```
void SetButtonSwitchedOn(string& asName, bool abSwitchedOn, bool abEffects);
```

Sticky areas

```
void SetAllowStickyAreaAttachment(bool abX);
```

Allows entites to stick to a StickyArea.

```
void AttachPropToStickyArea(string& asAreaName, string& asProp);
void AttachBodyToStickyArea(string& asAreaName, string& asBody);
```

Attaches a prop/body to a StickyArea.

```
void DetachFromStickyArea(string& asAreaName);
```

Detaches everything from a StickyArea.

Enemies

```
void SetNPCAwake(string& asName, bool abAwake, bool abEffects);
```

Activates the npc

```
void SetNPCFollowPlayer(string& asName, bool abX);
```

Sets an NPC's head to follow the player's movement's.

```
void SetEnemyDisabled(string& asName, bool abDisabled);
```

Disables an enemy.

```
void SetEnemyIsHallucination(string& asName, bool abX);
```

Makes an enemy a hallucination. Hallucinations fade to smoke when they get near the player.

```
void FadeEnemyToSmoke(string& asName, bool abPlaySound);
```

Instantly fades an enemy to smoke.

```
void ShowEnemyPlayerPosition(string& asName);
```

Makes the enemy run to the player, no matter where he is.

```
void AlertEnemyOfPlayerPresence(string &in asName);
```

Requires 1.3

Gives the specified enemy the player's current position and makes it search the area.

```
void SetEnemyDisableTriggers(string& asName, bool abX);
```

Enables or disables enemy triggers. If disabled, enemy will not react to player or attack.

```
void AddEnemyPatrolNode(string& asName, string& asNodeName, float
afWaitTime, string& asAnimation);
```

Adds a patrol node to the enemy's path.

asName - internal name of the enemy
asNodeName - path node

afWaitTime - time in seconds that the enemy waits at the path node before continuing
asAnimation - the animation the enemy uses when reaching the path node

```
void ClearEnemyPatrolNodes(string& asEnemyName);
```

Clears the current path of patrol nodes of the enemy.

```
void SetEnemySanityDecreaseActive(string &in asName, bool abX);
```

Requires 1.3

Enables/disables whether an enemy activates the player's sanity drain when stared at.

asName - Internal name of the enemy
abX - Enabled/disabled

```
void TeleportEnemyToNode(string &in asEnemyName, string &in asNodeName, bool
abChangeY);
```

Requires 1.3

Teleports an enemy to a specific PathNode.

asEnemyName - Internal name of the enemy
asNodeName - Internal name of the node to teleport to
abChangeY - Whether the Y position of the node will be used when teleporting the enemy

```
void TeleportEnemyToEntity(string &in asEnemyName, string &in
asTargetEntity, string &in asTargetBody, bool abChangeY);
```

Requires 1.3

Teleports an enemy to a specific entity.

asEnemyName - Internal name of the enemy
asTargetEntity - Internal name of the entity to teleport to
asTargetBody- Internal name of the entity's body name to teleport to. If empty, the first body will be used (might be unstable, recommended to input a body anyway)
abChangeY - Whether the Y position of the node will be used when teleporting the enemy

```
void ChangeManPigPose(string&in asName, string&in asPoseType);
```

Requires 1.3

Changes the pose a specified ManPig.

asName - Internal name of the enemy
asPoseType- Name of the ManPig pose to use. Can be “Biped” or “Quadruped”

```
void SetTeslaPigFadeDisabled(string&in asName, bool abX);
```

Requires 1.3

Enables/disables whether a specified TeslaPig should fade the player's view in and out.

asName - Internal name of the enemy
abX- Enabled/disabled

```
void SetTeslaPigSoundDisabled(string&in asName, bool abX);
```

Requires 1.3

Enables/disables whether a specified TeslaPig should play the proximity sounds.

asName - Internal name of the enemy
abX- Enabled/disabled

```
void SetTeslaPigEasyEscapeDisabled(string&in asName, bool abX);
```

Requires 1.3

Enables/disables whether a specified TeslaPig should be easier to escape from when hunted.

asName - Internal name of the enemy
abX- Enabled/disabled

```
void ForceTeslaPigSighting(string&in asName);
```

Requires 1.3

Forces a TeslaPig to be visible for a short time.

asName - Internal name of the enemy

```
string& GetEnemyStateName(string &in asName);
```

Requires 1.3

Returns the name of the state a specified enemy is current in. States can be Hunt, Search, Patrol, Wait, Alert, Investigate, Track and BreakDoor.

asName - Internal name of the enemy