6 Entity files

6.1 Intro

Entity files are XML files and before editing them it is suggested to read some quick tutorial on XML files. Only very basic XML understanding is needed to edit these files.

The files are divided into different sections, some sections are in all entity files and some are only present in certain entity types.

The following values are used, note that all types have “” around the value:

String	A string of letters. “MyName”
Float	A decimal number. “1.04”
Integer	A number without decimals “-15”
Vector	Three numbers in a row separated by space and/or commas: “1, 1, 1”.

6.1.1 Entity Types

Several different entity types have been created to simplify the creation of often used objects such as doors, lights and buttons. Much of the special entity functionality can be scripted in the .hps file for a level, but using special entities you can easily make an entity that can be imported to a level and have full functionality from the start. These entity types have extra settings mainly set in the <GAME/> section of the entity files. The specific settings for each entity type is described in Chapter 6.3.

Types are edited as described in Chapter 6.2.1.

The following entity types are available:

Type	Subtype	
Object	**Normal**	Regular objects with no specific function. Examples: Wooden box, rock
Enemy	**“EnemyName”**	Enemy entities, these have a very extended <GAME/> section. “EnemyName” = Worm, Dog, Spider. *This entity type is somewhat limited as to what sort of enemies have been in Penumbra.*
SwingDoor	**Normal**	Doors and objects with door like features. Examples: All regular doors that swing open, can also be used with success to create coffins, cupboards etc. A SwingDoor is easy to lock, unlock, make it breakable etc
Item	**Normal**	Item is an entity type that will be added to the players inventory. Normal is used for all regular items like keys, puzzle items and such. Basically an item that you want to be added to the inventory and that the player can then combine with another item, use on an item or some other basic item usage.
Item	**Battery**	Item is an entity type that will be added to the players inventory. Battery is added to the energy meter.

Item	**Dynamite**	Item is an entity type that will be added to the players inventory. Dynamite is is for... an object that will be thrown, explode and inflict damage.
Item	**Health**	Item is an entity type that will be added to the players inventory. Health is an item that can be “eaten”.
Item	**Flare**	Item is an entity type that will be added to the players inventory. Flare is an item that will be active in the players hand for a certain amount as configured in config/game.cfg.
Door	**Normal**	A door entity that contains animations, not used in the Tech Demo/Overture but is there and functional. Examples: Star Trek doors that slide open etc...
DoorPanel	**Normal**	A panel entity that is easily connectable to Door, enabling interacting with the panel to open/close door.
Lamp	**Normal**	Item is an entity type for creating an object with an easy to control light source. Examples: desk lamp, ceiling light, control panel with lights, a machine with lights.
Button	**Normal**	An entity that will easily control other entities. Examples: switch to turn on/off ceiling lights.
Lever	**Normal**	An entity that is a lever and as such can easily trigger events when pulled, can also be locked and unlocked.
Wheel	**Normal**	Not done, an entity that can be rotated several times and easily read out and trigger events based on that.

6.2 General sections

The following sections are present in most entity files used and are part of engine standards. Some extra options will also be mentioned that are not apart of the standard and game specific.

6.2.1 Main

This section exist in all entity files.

Name	The name of the entity type. *String*.
Type	The root type that is entity is. *String*.
Subtype	This is a subtype of the root type. For example the enemy type. *String*.

6.2.2 Physics

This section deals with the physical properties of an entity. For entities that have joints there are several versions of this section. One for each body connected with joints.

SubName	When entities with joints are used this is the name of the body. This should be the SHAPE name and not the node name! (i.e. pCube1Shape and not pCube1). *String*.
CollideCharacter	If the body collides with character bodies. *Boolean*.
Collides	If the entity colldies with the world. *Boolean*.
HasPhysics	If the physics are to be simulated. *Boolean*.
StaticMeshCollider	If a submesh should be used as a collider, mass must be 0. *Boolean*.
ColliderMesh	The name of the submesh used for StaticMeshCollider. This should be the SHAPE name and not the node name! (i.e. pCube1Shape and not pCube1). *String*.
Material	The name of the material used. *String*.
Mass	The mass of the body in kg. *Float*.
InertiaScale	Scale the inertia of the object (this is how easy it rotates). Should be “1 1 1” in most cases. *Vector*.
AngularDamping	Damping can be described has air friction. 1 is max friction and 0 is no fricton at all. Anglular means that this is the friction on the spinning motion of the entity, the more friction that faster it will stop spinning. Valid values are 0.0001 – 1.0, normally “0.1” is used. *Float*.
LinearDamping	The same as Angular damping but on the movement instead of spinning motion. The more friction the faster the entity will stop moving. *Float*.
BlocksSound	This body will make sound lower if the camera is behind it. *Boolean*.
HasGravity	If the body has gravity or not. *Boolean*.
MaxLinearSpeed	The maxium linear (up,down, right, left, etc) speed in m/s the body can have. 0 = unlimited speed. *Float*.
MaxAngularSpeed	The maxium angular (rotation) speed in m/s the body can have. 0 = unlimited speed. *Float*.
ContinuousCollision	If the body should use continuous collison detection or not. This is good for small bodies (or larger bodies that has one or many small colliders) that would easily stick or tunnel other bodies. It is also good for high speed bodies. *Boolean*.
PushedByCharacterGravity	If the body will be affected when a characters stands on top. *Boolean*.
PauseGravity	If gravity should be removed during interaction. *Boolean*.
AutoDisableLinearThreshold	Square of the min linear speed before body is disabled. Default 0.1 *Float*.
AutoDisableAngularThreshold	Square of the min angular speed before body is disabled. Default 0.1 *Float*.
AutoDisableNumSteps	Min steps of required speeds before the body is disabled. Default 10 *Int*.
Volatile	If a body might disappear or for some reason leave the way open for AI. Only needed on mass 0. Default false *Boolean*.
CanAttachCharacter	If a character standing on the body will be attached. For example if you have a moving platform and want the character to move with it as it moves. Default false *Boolean*.

6.2.3 Graphics

This section set the graphical appearance of the entities.

ModelFile	The name of the model file. No extension needed. *String.*
CastShadows	Sets if the entity casts shadows or not. *Boolean.*
AnimationLoop	If the start animation should be looped or not. *Boolean.*
AnimationRandStart	If the start animation should start at a random location, false starts always at 0 and true starts randomly between 0 and length of animation. *Boolean.*

6.2.4 Submesh

A submesh section is created for each submesh in the mesh.

Name	Name of the geometry for the submesh. *String*.
MaterialFile	Sets a new material for the submesh. If not the defined the one in the model is used. *String*.

6.2.5 Attach Billboards

The section contains info on how to attach billboards to lights, the styntax is as follows:

```
<ATTACH_BILLBOARDS>
    <Pair Light="rampspot1" Billboard="rampbb1" >
    [more pairs]
</ATTACH_BILLBOARDS>
```

Light	The name of the light to add the billboard to.
Billboard	The name of the billboard to be added.

6.2.6 Animation

The animation is used to give an object more than one animation. The model file defined by ModelFile in Graphics will be the main file (for more information, see 2.2). The syntax is the following for adding animations:

```
<ANIMATIONS>
    <Animation [properties]>
    […]
</ANIMATIONS>
```

The properties for each animation field are the following:

File	The model file that contains the animation. *String*.
Name	The name the animation will have ingame. *String*.
Speed	The speed of the animation in percent /100. 1 = normal speed, 0.5 = half speed, 2 = double speed. *Float*.
SpecialEventTime	Generic variable that is game dependant. Should signify some event happening in the animation *Float*.

Example:

```
<GRAPHICS
ModelFile = “test.dae”
/>
```

Test.dae will be the main file, it is in this file colliders and such must be.

```
<ANIMATIONS>
    <Animation File=”test_run.dae” Name=”Run” Speed=”1”>
    <Animation File=”test_jump.dae” Name=”Jump” Speed=”1”>
</ANIMATIONS>
```

This will add the animations “test_run.dae” and “test_jump.dae” and will name them “Run” and “Jump” in game. These files should not have colliders and such.

It's also possible to add events to an animation. Currently it's only used to add in sounds for an animation. See below example.

```
<ANIMATIONS>
    <Animation File=”test_run.dae” Name=”Run” Speed=”1” >
          <Event Time="0.1" Type="PlaySound" Value="name_of_snt_file"/>
       </Animation>
</ANIMATIONS>
```

The properties for each event field are the following:

Time	At what time in the animation to do the event. *String*.
Type	What type, currently on PlaySound is available. *String*.
Value	The name of the file to use for the event, for PlaySound that is what sound entity file to use. *String*.

6.2.7 Joint

Name	This is the name of the joint and is the same as the **name parameter**. If the joint is like this: **_joint_hinge_test_test_10_20_MyJoint** Then “MyJoint” is the name to use. *String*.

MinValue	Min limit value for joint, depends on joint type and overrides previous value if defined. *Integer*.
MaxValue	Max limit value for joint, depends on joint type and overrides previous value if define. *Integer*.
MaxLimit_Sound	The sound played when reaching the max limit. *String*.
MaxLimit_MinSpeed	The minimum speed for the sound to be played. *Float*.
MaxLimit_MaxSpeed	At this speed the sounds stops getting louder. *Float*.
MinLimit_Sound	The sound played when reaching the min limit. *String*.
MinLimit_MinSpeed	The minimum speed for the sound to be played. *Float*.
MinLimit_MaxSpeed	At this speed the sounds stops getting louder.*Float*.
MoveSound	The sound played when the joint objects are moving. *String*.
MoveType	The type of movement that the speed will be determined from, this can be “Linear” or “Angular”. This is up/down/left/etc movement and rotation respectively. *String*.
MinMoveSpeed	The minimum speed in m/s at which sound is heard. Valid values are 0 – infinity. *Float*.
MinMoveFreq	The lowest frequency played from the sound. Valid values are 0 – infinity. *Float*.
MinMoveVolume	The lowest volume played. . Valid values are 0 – 1. *Float*.
MinMoveFreqSpeed	The speed (and below) at which MinMoveFreq and MinMoveVolume is played. 0 – infinity. *Float*.
MaxMoveFreq	The highest frequency played from the sound. Valid values are 0 – infinity. *Float*.
MaxMoveVolume	The loudest volume played. . Valid values are 0 – 1. *Float*.
MaxMoveFreqSpeed	The speed (and above) at which MaxMoveFreq and MaxMoveVolume is played. 0 – infinity. *Float*.
MiddleMoveSpeed	The Speed at which frequency is 1 and MiddleMoveVolume. *Float*.
MiddleMoveVolume	The middle volume. *Float*.
Breakable	If the joint should break if enough force is applied to it. *Boolean*.
BreakForce	The minimum force for the joint to break. 0 – inf. *Float*.
BreakSound	The sound played when the joint breaks. *String*.
LimitAutoSleep	If there should be extra autosleep at limits. *Boolean*.
LimitAutoSleepDist	The distance at which the sleep occurs. 0 – inf. *Float*.
LimitAutoSleepNumSteps	The number of steps need for the sleep to start. 0 – inf. *Int*.
StickyMaxDistance	The distance from max limit that the body is stopped. 0 = disabled. (this is in radians for hinge). *Float*.
StickyMinDistance	The distance from min limit that the body is stopped. 0 = disabled. (this is in radians for hinge). *Float*.

6.2.8 Joint Controller

The Joint may also contain one or more Controllers. Controllers are used for controlling a body in some way. This maybe used to keep the body at a certain orientation or at a certain speed. This can simulate things like spinning fans and levers that go back to a certain position when released.

A demo of a Pid controller can be found in the Pid Demo folder.

Note that only the child body of the joint is affected by the controller, **not** the parent.

A controller is defined like this:

```
<Controller
[properties]
/>
```

And resides inside the joint tag, like:

```
<JOINT>
    <Controller />
<Controller />
    [continues for as many controllers are needed]
</JOINT>
```

At its basics a controller is a machine that gets an input and generates an output. The input could be the distance from a goal and the output could be a speed. This would create a controller that lowers the speed of an object as it gets closer to its goal. To control the amount of speed at a certain distance some constants are used. In most cases these the will make the relationship between input and output vary in a linear fashion, i.e. at half the distance the speed is at 50%, at 1/4th the distance 25% and so on.

There are two types of controllers. “Spring” and “Pid” these work in about the same way but can produce very different results.

Spring
This is probably the easiest type to understand. It is controlled by the formula:

Output = (DesiredValue – Input) * k – Input * b

The first part of the equation is used to control the output needed to reach the desired value. As you see the higher difference between desired value and input (called error) the higher output. The second part of the equation is used to make the output smaller if input gets too large. For example at a long distance you don’t want have too large speed. (Normally the same input is not used for the first and second part, but this is a simplified spring.)

Pid
A Pid is a little harder to describe than a spring. It consists of 3 constants; p, i and d. These are part of an equation but is better to explain what the 3 constants do instead of describing the equation used.

p	This is the proportional constant and works exactly like the first part of the spring formula. The higher it is, the higher the output generated will be. This has most influence over the generated output.
i	This is the integral term and can be said to be a memory of the previous errors. The greater the previous errors have been the greater affect this will have on the output. This is good to use if the input/output gets in a steady state (the output makes no change on the input) often, this state will make the old errors the same and thus it will increase the output by this term.

d	This is the derivative term and it is change in error. The term normally work negative on the output and is used to dampen the output when it is closing in too fast. The result will be that the desired valued is not over shot but instead reached in a smooth matter. The higher the term the more negative impact on the output when the error difference gets lower. This term will also give an extra boost to the output if the error difference gets higher (input gets further away from the desired value).

The properties are as follows:

Name	The name of the controller. *String*.
Active	If the controller is active or not. *Boolean*.
Type	The type of the controller. “Pid” or “Spring”.
A	p-term for Pid and k term for Spring. *Float*.
B	i-term for Pid and b term for Spring. *Float*.
C	d-term for Pid and not used in springs. *Float*.
IntegralSize	The number of errors saved for the integral term. 1 – inf. *Integer*.
Input	The type of input. “JointAngle”, “JointDist”,“LinearSpeed” or “AngularSpeed”.
InputAxis	The axis of the input (use “X” joint inputs). “X”,”Y” or “Z”.
DestValue	The desired value for input. *Float*.
Output	The output type. “Force” or “Torque”. (Torque is for rotation.)
OutputAxis	The axis of the output. “X”,”Y” or “Z”.
MaxOutput	The maximum output. 0 – infinity. 0 equals infinity (no upper limit). *Float*.
MulMassWithOutput	If the output should be multiplied with the body’s mass before being used. *Boolean*.
EndType	

This defines what is the end of the controller. When this happens the motor will become non active and another controller (if specified) will be active instead. This can be “Null”, “OnMax”, “OnMin” or “OnDest”.
Null: No end.
OnMax: The max limit of the joint is reached.
OnMin: The min limit of the joint is reached.
OnDest: DestValue is reached.

NextController	The controller that will be activated when the end (specified in EndType) of this controller is reached.
LogInfo	If the controller should write debug info to the log, default is false. *Boolean*.

6.2.9 Light

The light section can be used to change the properties to lights in a model, if section not present the values from the model will be used.

```
<LIGHT>
    Name = "pointLight1"
    […]
```

```
</LIGHT>
```

Name	The name of the light to change values for. *String*.
CastShadows	Should the light cast a shadow or not. *Boolean*.
Attenuation	The size of the light in meters. *Float*.
Color	The color of the light, “R G B A”. *Vector4*.
FOV	How quickly the radius of the light increases, as degree. **Only for spotlights.** *Float*.
Aspect	The difference between width and height increase set by the FOV, example: Aspect 1.5 for FOV 90 gives width=135 and height=90. **Only for spotlights.** *Float*.

6.3 Game Section

The game section of an entity file is the section that contains the most interesting properties, is in this section where the different types have all their specific settings.

The game section is defined like this:

```
<GAME
          InteractMode = “Static”
          [...] Other settings
/>
```

In it you add all the specifics for the different entity types as well as the standard features that are present in most entities. All settings are written **Setting = “The Value”**, it uses “ ” regardless of int, float, string, boolean.

6.3.1 Game Section General

These settings are usable in all entities, there might be some that only work for a certain type or some that work against each other. Overall should be no problem and nothing to risk by trial and error.

Name	Type	Default	Description
InteractMode	*String*	Static	Different modes for when you interact with the object, see below for details. “Static”, “Grab”, “Move”, “Push”.
ForwardUpMul	*Float*	1	MoveMode: Multiplied to the forward/backward force when moving the mouse up/down.
UpMul	*Float*	1	MoveMode: Multiplied to the up/down force when moving the mouse up/down.
RightMul	*Float*	1	MoveMode: Multiplied to the right/left force when moving the mouse right/left.
PickAtPoint	*Bool*	False	GrabMode: Should the object be picked that point the mouse is.

Name	Type	Default	Description
RotateWithPlayer	*Bool*	True	GrabMode: Should the object rotate along with the player.
UseNormalMass	*Bool*	False	GrabMode: Should the object's normal mass be used.
GrabMassMul	*Float*	1	GrabMode: Multiplied to the force applied to the object.
MaxInteractDist	*Float*	Depends	The maximum length at which the object can be interacted with.
PauseControllers	*Bool*		MoveMode: Should controllers be paused when interacting.
DamageCharacter	*Bool*	False	Should the characters take damage from the object.
MinLinearDamageSpeed	*Float*	0	Minimum linear speed at which damage is dealt.
MinAngularDamageSpeed	*Float*	0	Minimum angular speed at which damage is dealt.
MaxLinearDamageSpeed	*Float*	0	Maximum linear speed at which damage stops getting higher.
MaxAngularDamageSpeed	*Float*	0	Maximum angular speed at which damage stops getting higher.
MinDamage	*Float*	0	Damage dealt at minium speed.
MaxDamage	*Float*	0	Damage dealt at maximum speed.
DamageStrength	*???*	???	???.
Destroyable	*Bool*	false	If enemies can destroy all joints conneced to the body.
Toughness	*Int*	0	Thoughness of entity.
DestroyStrength	*Float*	0	Mimimum strength of the enemy to destroy the body.
DestroySound	*String*	“”	The sounds played when the object is destroyed.
LightFlash	*Bool*	“”	If a ligth flash should be displayed on destruction.
LightFlash_Color	*Color*	“R G B A”	Color of flash.
LightFlash_Radius	*Float*	0	Radius of flash.
LightFlash_AddTime	*Float*	0	The time it takes for falsh to expand.
LightFlash_NegTime	*Float*	0	The time it takes for falsh to contract.
LightFlash_Offset	*Vector3*	0	The offset from the object position.
CanBeThrown	*Bool*	True	Sets if an object can be thrown.
AttractEnemies	*Bool*	False	If the object will attract enemies to it.
AttractDistance	*Float*	0	The distance at which the attraction works.
AttractSubtypes	*String Vector*	“ ”	A string vec of the subtypes of enemies that are attraced. ie: “Dog Spider”.
AttractIsEaten	*Bool*	False	If the object is eaten once reached by the enemy.
AttractEatLength	*Float*	0	The time it takes for the object to be eaten.
CanBePulled	*Bool*	True	If an object with InteractMode Push can be pulled as well.
ForceLightOffset	*Bool*	False	If light should have an extra offset in absolute world coords.

Name	Type	Default	Description
LightOffset	*Vector3*	0 0 0	Extra light offest.
DisappearMinCloseDist	*Float*	0	When below this distance to another eneity of the same type, disappera instantly. 0 = off.

6.3.1.1 Additional specific information

Interact Modes	Description
Static	Nothing happens when you interact, example: ceiling light, large machine run by control panel.
Grab	Used to grab and move around objects, exampel: small rocks, bottles.
Move	Used to interact and move objects. These should be stuck onto something, example: a furniture door.
Push	Used to push around larger objects, example: crates, barrels.
MaxInteractDist	**Description**
Grab	Default value: 1.5.
Move	Default value: 1.4.
Push	Default value: 1.5.

6.3.2 Type Lamp

Name	Type	Default	Description
InteractDist	*Float*	???	Max distance you can interact with the lamp.
TurnOnTime	*Float*	1	Time for fade when turning on.
TurnOffTime	*Float*	1	Time for fade when turning off.
InteractOff	*Bool*	???	If you can interact to turn off.
InteractOn	*Bool*	???	If you can interact to turn on.
TurnOnSound	*String*	“ ”	Sound played when turning on.
TurnOffSound	*String*	“ ”	Sound played when turning off.
OnItem	*String*	“ ”	Item used to turn on, if “ ” then normal interaction works.
OffItem	*String*	“ ”	Item used to turn off, if “ ” then normal interaction works.
OffMaterial	*String*	“ ”	Material used when off.
OffSubMesh	*Float*	“ ”	Sub mesh to change material on when off.
OffColor	*Vector*	“R G B A”	The color of the light when off, 0 0 0 = black.
Flickering	*Bool*	False	Flickering active or not see the following options.
FlickerOffSound	*String*	“ ”	The sound to play when flickering off.
FlickerOnSound	*String*	“ ”	The sound to play when flickering on.
FlickerOffPS	*String*	“ ”	The particle effect to use when flickering off.
FlickerOnPS	*String*	“ ”	The particle effect to use when flickering on.
FlickerOnMinLength	*Float*	???	Minimum time the light will be on.
FlickerOnMaxLength	*Float*	???	Maximum time the light will be on.
FlickerOffMinLength	*Float*	???	Minimum time the light will be off.
FlickerOffMaxLength	*Float*	???	Maximum time the light will be off.
FlickerOffColor	*Vector*	“R G B A”	The color of the light when off, 0 0 0 = black.
FlickerOffRadius	*Float*	???	The radius of the light when off during a flicker.
FlickerFade	*Bool*	False	If it should fade between on and off.

Name	Type	Default	Description
FlickerOnFadeLength	*Float*	???	How long the fade should be when going to on.
FlickerOffFadeLength	*Float*	???	How long the fade should be when going to off.

6.3.3 Type SwingDoor

Entities with a door that swings open. There are some rules:
- May only have hinge joints
- Must open towards positive Z-axis.

Name	Type	Default	Description
Health	*float*	0	The health of the door, the door will be breakable this way.
Toughness	*int*	0	The toughness of the door, depending on toughness of the attacking entity the door will or will not take damage.

6.3.4 Type Door

Door is an old entity type that was added even before the tech demo. It's a door that uses animations to open and close, and when closed it can block portals making it a good optimization tool in crowded areas.

The Door entity needs 4 animations named: Opening, Open, Closing and Closed. It's an opening animation, the static open look, the closing animation and the static closed door. See Chapter 2.2 for mor details on animations.

```
    <ANIMATIONS>
        <Animation File="door_model_opening.dae" Name="Opening" Speed="1.0"
/>
        <Animation File="door_model_closed.dae"  Name="Closed" Speed="1.0"
/>
        <Animation File="door_model_closing.dae" Name="Closing" Speed="1.0"
/>
        <Animation File="door_model_open.dae" Name="Open" Speed="1.0" />
    </ANIMATIONS>
```

Name	Type	Default	Description
OpenStartSound	*String*	" "	Sound Played when started opening.
OpenLoopSound	*String*	" "	Sound Looped when opening.
OpenStopSound	*String*	" "	Sound Played when stopped opening.
OpenLoopStartFadeSpeed	*Float*	1	Fade in speed of loop sound.
OpenLoopStopFadeSpeed	*Float*	1	Fade out speed of loop sound.
CloseStartSound	*String*	" "	Sound Played when started closing.
CloseLoopSound	*String*	" "	Sound Looped when closing.
CloseStopSound	*String*	" "	Sound Played when stopped closing.

Name	Type	Default	Description
CloseLoopStartFadeSpeed	*Float*	1	Fade in speed of loop sound.
CloseLoopStopFadeSpeed	*Float*	1	Fade out speed of loop sound.
OpenWhenBlocked	*Bool*	False	If the door should openitself if blocked. For example, place a box in the gap to bounce doors open on collide.
BlockPortals	*Bool*	True	If the door blocks portals when closed.

6.3.5 Type DoorPanel

An Entity that is an object that the player can interact with to trigger the Door entity. They require a certain naming that needs to be looked into, it's a sort of lost knowledge as it has never been used beyond early testing...

Name	Type	Default	Description
ChangeWhileMoving	*Bool*	False	If the door can have its state changed while it is moving.

6.3.6 Type Button

Name	Type	Default	Description
InteractDist	*Float*	???	Max interaction distance.
InteractOff	*Bool*	???	If you can turn off by interacting.
InteractOn	*Bool*	???	If you can turn on by interacting.
TurnOnSound	*String*	“ ”	Sound played when turning on.
TurnOffSound	*String*	“ ”	Sound played when turning off.
TurnOnAnimation	*String*	“ ”	Animation played when turning on.
TurnOffAnimation	*String*	“ ”	Animation played when turning off.
OffMaterialName	*String*	“ ”	Material used when turned off.
OffSubMesh	*String*	“ ”	Sub mesh that material is switch on (if OffMaterialName != “”).

6.3.7 Type Wheel

Name	Type	Default	Description
InteractDist	*Float*	1.8f	Max interaction distance.
MinLimit	*Float*	0	Min limit of wheel (should be negative).
MaxLimit	*Float*	0	Max limit of wheel (should be postive).
MinSound	*String*	“ ”	Sound played at min.
MaxSound	*String*	“ ”	Sound played at max.
StuckAtMin	*Bool*	false	If it gets stuck at min.
StuckAtMax	*Bool*	false	If it gets stuck at max.
SpinDir	*int*	0	0=both dirs, 1=to max, -1=to min.
PauseControllers	*bool*	true	if controllers should be paused during player interaction.

6.3.8 Type Enemy

6.3.8.1 Enemy Dog

An enemy that can take care of itself.

Needs animations for:

- Standing still, named: Idle
- Walking forwards, named: Walk
- Walking backwards, named: Backward
- Running, named: Run
- Eating, named: Eating
- Attacking high, named: Attack
- Rise right after knocked, named: RiseRight
- Rise left after knocked,, named: RiseLeft
- Attacking a door to break it, named: BreakDoor
- Calling for backup, named: Howl
- Attacking low, named: AttackLow
- Seeing player and waiting to attack, named:Angry

Name	Type	Description
ShowDebug	*Bool*	Displays additional informaiton on the screen about the enemy and it's behavior, useful to check path nodes etc.
Disappear	*Bool*	If the enemy should disappear on death.
DisappearMinTime	*Float*	The minimum of time before disappearing.
DisappearMaxTime	*Float*	The maximum of time before disappearing.
DisappearMinDistance	*Float*	The minimum of distance from player to disappear.
DisappearPS	*String*	Particle effect to display on disappearing.
DisappearSound	*String*	Sound to play on disappearing.
DisappearFreezesRagdoll	*Bool*	If the Ragdoll should turn static or remain active on diseappering.
CloseMusic	*String*	The music to play when enemy close to player.
CloseMusicPrio	*Int*	The priority of the music, should be higher than ambient music of level or else it wont play.
CloseMusicStartDist	*Float*	At what distance the music should start playing.
CloseMusicStopDist	*Float*	At what distance the music should stop playing.
AttackMusic	*String*	The music to play when enemy enters attack mode.
AttackMusicPrio	*Int*	The priority of the music, should be higher than ambient music of level or else it wont play.
MaxPushMass	*Float*	The maximum mass of an object that the enemy can move.
PushForce	*Float*	At what force an emeny pushes an object.

Name	Type	Description
MaxHealth	*Float*	The maximum health on an enemy, regenerates other time. 100 = players health.
MaxSeeDist	*Float*	The maximum distance that the enemy can spot the player.
MaxForwardSpeed	*Float*	Maximum forward speed of enemy, default 1.
Acceleration	*Float*	Acceleration speed.
Deacceleration	*Float*	Deceleration speed.
StoppedToWalkSpeed	*Float*	How long to go from halt to walk.
WalkToStoppedSpeed	*Float*	How long to go from walk to halt.
WalkToRunSpeed	*Float*	How long to go from walk to run.
RunToWalkSpeed	*Float*	How long to go from run to walk.
MoveAnimSpeedMul	*Float*	Speed multiplier for the animations, to sync the animation to the movement speed.
BodySize	*Vector3*	The size of the body.
BodyMass	*Float*	The mass of the body.
ModelOffset_Rot	*Vector3*	Rotating offset for the model.
ModelOffset_Pos	*Vector3*	Position offset for the model.
AlignToGroundNormal	*Bool*	If the body should align to the ground as the ground changes angel.
HitPS	*String*	Particle effect to be played at point of impact on enemy.
MinKnockDamage	*Float*	The minimum damage where the enemy might be knocked over.
CertainKnockDamage	*Float*	The amount of damage where the enemy will always be knocked over.
LengthBodyToAss	*Float*	Who knows...
IdleFOV	*Float*	Field of view for enemy when idle.
IdleFoundPlayerSound	*String*	Sound for enemy spotting the player.
IdleMinSeeChance	*Float*	Minimum chance of enemy seeing player. 0 never 1 always, 0.x the chance.
IdleMinHearVolume	*Float*	Minimum chance of enemy hearing the player. 0 never 1 always, 0.x the chance.
IdleSound	*String*	Sound to be played at random intervals.
IdleSoundMinInteraval	*Float*	Minimum time between each time it plays the idle sound.
IdleSoundMaxInteraval	*Float*	Maximum time between each time it plays the idle sound.
IdleCallBackupChance	*Float*	Chance of the enemy running away and calling for backup, 0 never 1 always, 0.x the chance.
InvestigateSound	*String*	Sound to be played when enemy hears or sees something worth investigating.
AttentionSound	*String*	Sound to be played when the enemy sees player and goes into attention.
AttentionTime	*Float*	For how long the enemy will stay in attention, if player moves out of sight before end enemy will go back.
AttentionMinDist	*Float*	Minimum distance for attention to happen, if enemy within this distance it will go to attack directly.
HuntFOV	*Float*	Field of view for enemy when in hunt mode.
HuntSpeed	*Float*	At what speed the enemy moves when in hunt mode.

Name	Type	Description
HuntForLostPlayerTime	*Float*	For how long will enemy continue to hunt for the player when enemy lost track of player.
HuntMinSeeChance	*Float*	Minimum chance of seeing the player when in hunt. 0 never 1 always, 0.x the chance.
HuntMinHearVolume	*Float*	Minimum chance of hearing the player when in hunt. 0 never 1 always, 0.x the chance.
AttackDistance	*Float*	At what distance the enemy will do it's attack.
AttackSpeed	*Float*	At what speed the attack will happen.
AttackJumpTime	*Float*	How long to delay before jump.
AttackDamageTime	*Float*	How long the damage from the attack will be active.
AttackDamageSize	*Vector3*	How large the area will be affected by the attack.
AttackDamageRange	*Float*	How far the attack will reach.
AttackMinDamage	*Float*	Minimum damage dealt with each attack.
AttackMaxDamage	*Float*	Maximum damage dealt with each attack.
AttackStartSound	*String*	The sound to play at the beginning of an attack.
AttackHitSound	*String*	The sound to play when the attack is successful.
AttackMinMass	*Float*	Minimum mass that will be affected by an attack.
AttackMaxMass	*Float*	Maximum mass that will be affected by an attack.
AttackMinImpulse	*Float*	Minimum impulse given from an attack.
AttackMaxImpulse	*Float*	Maximum impulse given from an attack.
AttackStrength	*Float*	The strength of an attack.
BreakDoorAnimation	*String*	The animation to use for enemy attacking a door it encounters locked/blocked.
BreakDoorSpeed	*Float*	At what speed to play the animation.
BreakDoorDamageTime	*Float*	How long the damage will be active for the attack.
BreakDoorDamageSize	*Vector3*	The size of the area affected by the attack.
BreakDoorDamageRange	*Float*	The range of the attack.
BreakDoorMinDamage	*Float*	Minimum damage dealt with attack.
BreakDoorMaxDamage	*Float*	Maximum damage dealt with attack.
BreakDoorStartSound	*String*	Sound of start of door attack.
BreakDoorHitSound	*String*	Sound when hitting the door during attack.
BreakDoorMinMass	*Float*	The minimum mass affected by the attack.
BreakDoorMaxMass	*Float*	The maximum mass affected by the attack.
BreakDoorMinImpulse	*Float*	The minimum impulse given by the attack.
BreakDoorMaxImpulse	*Float*	The maximum impulse given by the attack.
BreakDoorStrength	*Float*	The strength of the attack on the door,.
BreakDoorRiseAtEnd	*Bool*	Should the rise animation be played at the end of the break door animation.
KnockDownSound	*String*	The sound played as an enemy is knocked over.
DeathSound	*String*	The sound played as an enemy dies.
FleePositionChance	*Float*	The chance of fleeing when knocked. 0 never 1 always, 0.x the chance.
FleePositionMaxTime	*Float*	How long the flee will last.
FleePositionMinDistance	*Float*	Minimum distance to run away.

Name	Type	Description
FleePositionMaxDistance	*Float*	Maximum distance to run away.
FleeBackChance	*Float*	The chance of running back to the player. 0 never 1 always, 0.x the chance.
FleeBackTime	*Float*	How long it will wait to run back.
FleeBackSpeed	*Float*	At what speed to run back.
CallBackupAnimation	*String*	What animation to use when calling for backup.
CallBackupSound	*String*	What sound to play during call for backup animation.
CallBackupRange	*Float*	How far the backup call will reach, one enemy within the range will come to help the other.
EatFOV	*Float*	The field of view while eating
EatMinSeeChance	*Float*	The chance it will see the player while eating. 0 never 1 always, 0.x the chance.
EatMinHearVolume	*Float*	The chance it will hear the player while eating. 0 never 1 always, 0.x the chance.

6.3.8.2 Enemy Spider

A erratic enemy that hunts the player when seen, and optionally runs away from the flashlight.

Needs animations for:

- Standing still, named: Idle
- Walking forwards, named: Walk
- Running, named: Run
- Attacking high, named: Attack
- Death animation that takes over after landing from ragdoll, named:Death

Name	Type	Description
ShowDebug	*Bool*	Displays additional informaiton on the screen about the enemy and it's behavior, useful to check path nodes etc.
Disappear	*Bool*	If the enemy should disappear on death.
DisappearMinTime	*Float*	The minimum of time before disappearing.
DisappearMaxTime	*Float*	The maximum of time before disappearing.
DisappearMinDistance	*Float*	The minimum of distance from player to disappear.
DisappearPS	*String*	Particle effect to display on disappearing.
DisappearSound	*String*	Sound to play on disappearing.
DisappearFreezesRagdoll	*Bool*	If the Ragdoll should turn static or remain active on diseappering.
CloseMusic	*String*	The music to play when enemy close to player.
CloseMusicPrio	*Int*	The priority of the music, should be higher than ambient music of level or else it wont play.
CloseMusicStartDist	*Float*	At what distance the music should start playing.
CloseMusicStopDist	*Float*	At what distance the music should stop playing.
AttackMusic	*String*	The music to play when enemy enters attack mode.

Name	Type	Description
AttackMusicPrio	*Int*	The priority of the music, should be higher than ambient music of level or else it wont play.
MaxPushMass	*Float*	The maximum mass of an object that the enemy can move.
PushForce	*Float*	At what force an emeny pushes an object.
MaxHealth	*Float*	The maximum health on an enemy, regenerates other time. 100 = players health.
MaxSeeDist	*Float*	The maximum distance that the enemy can spot the player.
AlignToGroundNormal	*Bool*	If the body should align to the ground as the ground changes angel.
MaxForwardSpeed	*Float*	Maximum forward speed of enemy, default 1.
Acceleration	*Float*	Acceleration speed.
Deacceleration	*Float*	Deceleration speed.
mfMaxTurnSpeed	*Float*	Turning speed.
mfAngleDistTurnMul	*Float*	Multiplier for turning ???.
mfMinBreakAngle	*Float*	Break angle ???
mfBreakAngleMul	*Float*	Multiplier for break angel ???
StoppedToWalkSpeed	*Float*	How long to go from halt to walk.
WalkToStoppedSpeed	*Float*	How long to go from walk to halt.
WalkToRunSpeed	*Float*	How long to go from walk to run.
RunToWalkSpeed	*Float*	How long to go from run to walk.
MoveAnimSpeedMul	*Float*	Speed multiplier for the animations, to sync the animation to the movement speed.
BodySize	*Vector3*	The size of the body.
BodyMass	*Float*	The mass of the body.
ModelOffset_Rot	*Vector3*	Rotating offset for the model.
ModelOffset_Pos	*Vector3*	Position offset for the model.
HitPS	*String*	Particle effect to be played at point of impact on enemy.
MinKnockDamage	*Float*	The minimum damage where the enemy might be knocked over.
CertainKnockDamage	*Float*	The amount of damage where the enemy will always be knocked over.
LengthBodyToAss	*Float*	Who knows...
IdleFOV	*Float*	Field of view for enemy when idle.
IdleFoundPlayerSound	*String*	Sound for enemy spotting the player.
IdleMinSeeChance	*Float*	Minimum chance of enemy seeing player. 0 never 1 always, 0.x the chance.
IdleMinHearVolume	*Float*	Minimum chance of enemy hearing the player. 0 never 1 always, 0.x the chance.
IdleMinWaitLength	*Float*	Minimum time it stands still waiting.
IdleMaxWaitLength	*Float*	Maximum time it stands still waiting.
HuntFOV	*Float*	Field of view for enemy when in hunt mode.
HuntSpeed	*Float*	At what speed the enemy moves when in hunt mode.

Name	Type	Description
HuntForLostPlayerTime	*Float*	For how long will enemy continue to hunt for the player when enemy lost track of player.
HuntMinSeeChance	*Float*	Minimum chance of seeing the player when in hunt. 0 never 1 always, 0.x the chance.
HuntMinHearVolume	*Float*	Minimum chance of hearing the player when in hunt. 0 never 1 always, 0.x the chance.
AttackDistance	*Float*	At what distance the enemy will do it's attack.
AttackForce	*Float*	At what what force to do the attack.
AttackJumpTime	*Float*	How long to delay before jump.
AttackDamageTime	*Float*	How long the damage from the attack will be active.
AttackDamageSize	*Vector3*	How large the area will be affected by the attack.
AttackDamageRange	*Float*	How far the attack will reach.
AttackMinDamage	*Float*	Minimum damage dealt with each attack.
AttackMaxDamage	*Float*	Maximum damage dealt with each attack.
AttackStartSound	*String*	The sound to play at the beginning of an attack.
AttackHitSound	*String*	The sound to play when the attack is successful.
AttackMinMass	*Float*	Minimum mass that will be affected by an attack.
AttackMaxMass	*Float*	Maximum mass that will be affected by an attack.
AttackMinImpulse	*Float*	Minimum impulse given from an attack.
AttackMaxImpulse	*Float*	Maximum impulse given from an attack.
AttackStrength	*Float*	The strength of an attack.
KnockDownSound	*String*	The sound played as an enemy is knocked over.
DeathSound	*String*	The sound played as an enemy dies.
FleeMinDistance	*Float*	Minimum distance the enemy runs away when fleeing.
FleeMaxDistance	*Float*	Maximum distance the enemy runs away when fleeing.
FleeFromFlashlight	*Float*	IF the enemy should be scared of the flashlight beam.

6.3.8.3 Enemy Worm

An enemy that follows the player around and deals damage at set intervals.

Needs animations for:

- Being still, named: Idle
- Moving, named: Move
- Attacking, named: Attack

Name	Type	Description
ShowDebug	*Bool*	Displays additional informaiton on the screen about the enemy and it's behavior, useful to check path nodes etc.
Disappear	*Bool*	If the enemy should disappear on death.
CloseMusic	*String*	The music to play when enemy close to player.

Name	Type	Description
CloseMusicPrio	*Int*	The priority of the music, should be higher than ambient music of level or else it wont play.
CloseMusicStartDist	*Float*	At what distance the music should start playing.
CloseMusicStopDist	*Float*	At what distance the music should stop playing.
AttackMusic	*String*	The music to play when enemy enters attack mode.
AttackMusicPrio	*Int*	The priority of the music, should be higher than ambient music of level or else it wont play.
MaxPushMass	*Float*	The maximum mass of an object that the enemy can move.
PushForce	*Float*	At what force an emeny pushes an object.
MaxHealth	*Float*	The maximum health on an enemy, regenerates other time. 100 = players health.
MaxSeeDist	*Float*	The maximum distance that the enemy can spot the player.
AlignToGroundNormal	*Bool*	If the body should align to the ground as the ground changes angel.
MaxForwardSpeed	*Float*	Maximum forward speed of enemy, default 1.
Acceleration	*Float*	Acceleration speed.
Deacceleration	*Float*	Deceleration speed.
mfMaxTurnSpeed	*Float*	Turning speed.
mfAngleDistTurnMul	*Float*	Multiplier for turning ???.
mfMinBreakAngle	*Float*	Break angle ???
mfBreakAngleMul	*Float*	Multiplier for break angel ???
StoppedToWalkSpeed	*Float*	How long to go from halt to walk.
WalkToStoppedSpeed	*Float*	How long to go from walk to halt.
WalkToRunSpeed	*Float*	How long to go from walk to run.
RunToWalkSpeed	*Float*	How long to go from run to walk.
MoveAnimSpeedMul	*Float*	Speed multiplier for the animations, to sync the animation to the movement speed.
BodySize	*Vector3*	The size of the body.
BodyMass	*Float*	The mass of the body.
ModelOffset_Rot	*Vector3*	Rotating offset for the model.
ModelOffset_Pos	*Vector3*	Position offset for the model.
HitPS	*String*	Particle effect to be played at point of impact on enemy.
MoveSound	*String*	A sound to play while moving.
IdleFOV	*Float*	Field of view for enemy when idle.
IdleFoundPlayerSound	*String*	Sound for enemy spotting the player.
IdleMinSeeChance	*Float*	Minimum chance of enemy seeing player. 0 never 1 always, 0.x the chance.
IdleMinHearVolume	*Float*	Minimum chance of enemy hearing the player. 0 never 1 always, 0.x the chance.
IdleMinWaitLength	*Float*	Minimum time it stands still waiting.
IdleMaxWaitLength	*Float*	Maximum time it stands still waiting.
HuntFOV	*Float*	Field of view for enemy when in hunt mode.
HuntSpeed	*Float*	At what speed the enemy moves when in hunt mode.

Name	Type	Description
HuntForLostPlayerTime	*Float*	For how long will enemy continue to hunt for the player when enemy lost track of player.
HuntMinSeeChance	*Float*	Minimum chance of seeing the player when in hunt. 0 never 1 always, 0.x the chance.
HuntMinHearVolume	*Float*	Minimum chance of hearing the player when in hunt. 0 never 1 always, 0.x the chance.
AttackInterval	*Float*	At what intervals the attacks will be.
AttackDamage	*Float*	The damage dealt from an attack.
AttackHitSound	*String*	The sound to play when attacking with a hit.
AttackHitSoundInterval	*Float*	The interval between hit sounds.
AttackMinMass	*Float*	Minimum mass that will be affected by an attack.
AttackMaxMass	*Float*	Maximum mass that will be affected by an attack.
AttackMinImpulse	*Float*	Minimum impulse given from an attack.
AttackMaxImpulse	*Float*	Maximum impulse given from an attack.
AttackStrength	*Float*	The strength of an attack.
AttackDamageSize	*Vector3*	How large the area will be affected by the attack.

6.3.8.4 Enemy Roach (Flying dolphin)

All properties hardcoded. ???

6.3.9 Type Item

Name	Type	Default	Description
ImageFile	*String*	" "	The image file to be loaded.
CanBeDropped	*Bool*	True	If it should be possible to drop the item.
NameCat	*String*	" "	Translation Category for name.
NameEntry	*String*	" "	Translation Entry for name.
DescCat	*String*	" "	Translation Category for description.
DescEntry	*String*	" "	Translation Entry for description.
ItemType	*String*	"Normal"	Type of Item, see below for possible values.
HasCount	*Bool*	False	If the item has a count, ie it only takes up one slot.
Count	*Int*	1	If the items has a count this specifies how much the count increases.
PickUpSound	*String*	"player_pickup_generic"	The sound that is played when the item is picked.
HudModelFile	*String*	" "	The hud file to be used.
HudModelName	*String*	" "	The name of the object in the hud file.
EnterFlashDist	*Float*	3	The Distance at which an item can flash.
ExitFlashDist	*Float*	6	The distance at which the flash will be reset and it might flash again (when in sight and near enough).
SkipRayCheck	*Bool*	False	Skips the ray check when doing flashes.

6.3.9.1 Different types of Items

ItemType	Description
Normal	Regular items, as it's default you do not need to add an ItemType to the GAME unless you are going to specify any of the below.
Note	Note item that will be added to notebook, is a Type="Item", SubType="Normal".
Map	Map item that will be added to map folder, this type was canceled but might still be functional.
Battery	Battery item, is a Type="Item", SubType="Battery".
Throw	A throwable object like the dynamite that can be wielded and charged to throw, is a Type="Item", SubType="Dynamite"/SubType="DogFood" .
Flashlight	Flashlight item that will run on batteries, is a Type="Item", SubType="Normal".
GlowStick	Light item that will last indefinitely, is a Type="Item", SubType="Normal".
Flare	Light item that will burn out, is a Type="Item", SubType="Flare".
Food	Eatable, do not think it has been used at all. But it should be there.. maybe.
Painkillers	A health item that restores the players health, is a Type="Item", SubType="Health".
WeaponMelee	An item that can be wielded and swung around, is a Type="Item", SubType="Normal".